Deeply Tensor Compressed Transformers for End-to-End Object Detection

Peining Zhen, Ziyang Gao, Tianshu Hou, Yuan Cheng, and Hai-Bao Chen*

Shanghai Jiao Tong University, China
{zhenpn, gaoziyang, houtianshu, cyuan328, haibaochen}@sjtu.edu.cn

Abstract

DEtection TRansformer (DETR) is a recently proposed method that streamlines the detection pipeline and achieves competitive results against two-stage detectors such as Faster-RCNN. The DETR models get rid of complex anchor generation and post-processing procedures thereby making the detection pipeline more intuitive. However, the numerous redundant parameters in transformers make the DETR models computation and storage intensive, which seriously hinder them to be deployed on the resources-constrained devices. In this paper, to obtain a compact end-to-end detection framework, we propose to deeply compress the transformers with low-rank tensor decomposition. The basic idea of the tensor-based compression is to represent the large-scale weight matrix in one network layer with a chain of low-order matrices. Furthermore, we propose a gated multi-head attention (GMHA) module to mitigate the accuracy drop of the tensor-compressed DETR models. In GMHA, each attention head has an independent gate to determine the passed attention value. The redundant attention information can be suppressed by adopting the normalized gates. Lastly, to obtain fully compressed DETR models, a low-bitwidth quantization technique is introduced for further reducing the model storage size. Based on the proposed methods, we can achieve significant parameter and model size reduction while maintaining high detection performance. We conduct extensive experiments on the COCO dataset to validate the effectiveness of our tensor-compressed (tensorized) DETR models. The experimental results show that we can attain 3.7× full model compression with 482× feed forward network (FFN) parameter reduction and only 0.6 points accuracy drop.

Introduction

Recent years have witnessed great success in object detection area with the rapid development of deep learning and neural networks (Ren et al. 2015; Lin et al. 2017; Carion et al. 2020). Numerous detectors have been proposed to continually push forward the state-of-the-art. Nowadays, anchor-based and anchor-free detectors are the dominant methods for efficient target object detection. These methods make bounding box (bbox) predictions depending on proposals (Cai and Vasconcelos 2018), anchors (Lin et al. 2017), or key points (Zhou, Wang, and Krähenbühl 2019; Law and Deng 2018). However, the design and generation of the above prior knowledge make the corresponding network structures more complex. Currently, another branch of object detection methods beyond anchor-based ones has drawn wide attention (Carion et al. 2020; Zhu et al. 2020; Dai et al. 2021). These methods are inspired by the widely used transformer architecture (Vaswani et al. 2017) and streamline the detection pipeline.

The detection transformer (DETR) (Carion et al. 2020), which achieves competitive results against strong baselines, is the first attempt to simplify the detection pipeline based on transformers. However, the vanilla DETR models suffer from large model size and high computational cost since most of their network structures are linear layers. Such high demands for computing resources seriously hinder them to be stored and deployed on mobile devices with limited hardware resources and a tight power budget. It thereby remains a great challenge to develop compact DETR models while maintaining the simplicity of their detection pipeline. Although the works (Zhu et al. 2020; Sun et al. 2021; Dai et al. 2021) make efforts to improve the training efficiency of DETR models, they do not focus on reducing the model size. For resources-constrained AI applications, such as self-driving cars and AR glasses, model compression is particularly necessary.

It is intuitive to develop model compression methods for reducing the complexity of neural networks. One of the major compression methods is quantization-aware training (QAT), which aims to reduce the bitwidth of network weights and activations during training based on pre-trained models (Han, Mao, and Dally 2015; Jacob et al. 2018). Nevertheless, these QAT methods are tedious and time-consuming, which hinder the immediate application of quantization techniques.f Downstream toolchains (e.g., ONNX or TensorFlow-Lite) are required to accomplish the model size reduction after QAT. Another main compression method is network pruning (He, Zhang, and Sun 2017; Zhuang et al. 2018), which focuses on removing the redundant network structure. However, the pruning metric is difficult to decide and the retraining procedure is complicated. In this work, we propose to compress the DETR models through low-rank tensor decomposition. The neural network weights can be represented as a combination of mul-

*Corresponding Author

tiple very small tensor arrays. Consequently, the number of required trainable parameters can be significantly reduced. Different from QAT and pruning methods, the tensor-based compression methods directly reconstruct the network structure and the parameters can be learned from scratch. As a result, the expensive computational costs in the retraining procedure can be avoided, and high detection accuracy can remain.

In this paper, we propose a tensorized compression framework for the detection transformer. By reshaping the weight matrices into high-dimensional tensors and adopting a low-rank tensor factorization, significant redundant parameter reduction can be achieved with maintained detection accuracy. Furthermore, to alleviate the accuracy drop caused by the tensorized compression, we propose a gated multi-head attention (GMHA) module. Learnable gates are attached to every single attention head for spotlighting meaningful heads and down-weighting uninformative ones. At last, a post-training quantization technique is introduced for further model size reduction. For the COCO detection benchmark, the tensorized DETR shows 3.7× model compression with 482× FFN parameter reduction and only 0.6% accuracy drop (39.5% vs. 40.1%). The experimental results and discussions can demonstrate the advantages of our proposed method.

The highlights of this paper can be summarized as follows: **1)** We first propose to compress the detection transformer based on the tensor decomposition. The storage-intensive structures such as FFNs can be reconstructed in tensor format to directly achieve parameter reduction. This method does not need any downstream toolchains or backends to realize model size compression; **2)** A gated multi-head attention module is proposed to mask the redundant heads in the transformer architectures. The controllable gate parameters are independently learned during training rather than manually fixed. The accuracy drop after compression can be mitigated with limited parameters increase; **3)** A post-training quantization technique is introduced to further compress the model size. This method preserves high model performance without any tedious model retraining procedure; **4)** Extensive results on the COCO dataset can demonstrate the advantages of our proposed methods. The DETR model is deeply compressed with competitive accuracy against state-of-the-art approaches.

Related Work

Object Detection

Modern object detection methods can be mainly divided into two categories: anchor-based and anchor-free methods. The anchor-based methods (Redmon and Farhadi 2017; He et al. 2017) make predictions with the pre-defined anchor boxes, and the quality of anchors is proven to have a significant influence on the final performance (Zhang et al. 2020). In general, the anchor-based methods have higher accuracy compared with anchor-free methods since the positive and negative samples are more balanced. However, the design and generation of anchors will greatly increase the model complexity, therefore the inference speed and model size of anchor-based detectors are far from satisfactory. Most anchor-free methods (Law and Deng 2018; Tian et al. 2019) regress bounding boxes with the extracted keypoints. The anchor-free methods remove heavy anchor generation networks and post-processing procedures thus they can easily achieve high speed and lightweight while maintaining high accuracy.

Recently, many works have been proposed to improve the efficiency of DETR-series models (Zhu et al. 2020; Dai et al. 2021; Zheng et al. 2020; Sun et al. 2021). In the work (Zhu et al. 2020), the authors propose multi-scale deformable attention with learnable sparse sampling for boosting the detection performance without increasing redundant parameters. UP-DETR (Dai et al. 2021) involves the unsupervised pretraining mechanism into DETR for fast convergence and high performance. The adaptive clustering transformer (ACT) (Zheng et al. 2020) proposes to cluster the query features adaptively using locality sensitive hashing (LSH) for reducing the inference computation cost. TSP-RCNN and TSP-FCOS are proposed based on the transformer set prediction (TSP) in (Sun et al. 2021) and can achieve faster convergence speed with better detection performance. The above methods mainly focus on improving the training efficiency of DETR models while our method is devoted to reducing the DETR model size for hardware-friendly deployment on resources-constrained devices.

Neural Network Compression

Researchers have proposed a variety of neural network compression methods. One of the major compression methods is quantization, which narrows the bitwidth of network weights or activations to limit the model size. Recent research efforts such as (Han, Mao, and Dally 2015; Hubara et al. 2016; Rastegari et al. 2016) have significantly reduced the computational complexity and model size by adopting low-bitwidth weights and activations. Specifically, in BNN (2016) and XNOR-Net (2016), the network parameters are aggressively quantized into 2 bits with negligible accuracy loss. Besides the above approaches with fixed bitwidth, the works (Khoram and Li 2018; Wang et al. 2019) develop flexible quantization policies. The network layers have different bitwidths during training; nevertheless, they obtain similar accuracy compared with fixed bitwidth methods. In summary, the above methods belong to quantization-aware training which require full training data and a time-consuming retraining procedure.

Recently, the studies (Li, Wang, and Kong 2018; Gusak et al. 2019; Yin et al. 2021) propose to compress neural networks via tucker tensor decomposition methods. In (Li, Wang, and Kong 2018; Yin et al. 2021), the authors adopt tucker based low-rank approximation to compress the neural networks for classification tasks. And in (Li, Wang, and Kong 2018), the compressed GoogLeNet can achieve about 2× compression on an ARM-based cell phone. In (Gusak et al. 2019), the authors leverage tucker decomposition to compress object detection models in an iterative procedure. The compressed Faster-RCNN can achieve up to 3.16× parameters reduction with negligible detection accuracy loss. Besides, in (Wang et al. 2018), the authors can compress the network up to 243× without losing accuracy. However, the authors mainly focus on shallow networks and simple tasks; they do not provide the results on more difficult tasks such as detection or segmentation. In our work, the transformer weights are factorized into a sequence of small tensors and we can achieve a much higher compression ratio.

The Proposed Method

Notations and The Basis of Tensor A tensor-based compression method with open boundary conditions, namely tensor-train decomposition, is introduced in this paper for further optimizing the DETR model. We limit our study to only change FFN weights in this section.

In general, tensors are multi-dimensional arrays that can generalize vectors and matrices. We refer the vectors and matrices as 1-dimensional arrays and 2-dimensional arrays respectively. In this section, the vectors are denoted by ordinary lower case letters (e.g. v); the matrices are denoted by boldface upper case letters (e.g. $\mathbf{V}$); the tensors are denoted by calligraphic upper case letters (e.g. $\mathcal{V}$). A tensor slice is a 2-dimensional fragment of one tensor, obtained by fixing two of all indices. For example, given a 3-dimensional tensor $\mathcal{V} \in \mathbb{R}^{I\times J\times K}$, the horizontal, lateral, and frontal slices of this tensor can be represented as $\mathcal{V}(i,:,:)$, $\mathcal{V}(:,j,:)$, and $\mathcal{V}(:,:,k)$ respectively.

Tensor-Train Preliminaries Via the tensor-train decomposition, a high-order tensor can be expressed as a chain of matrix products. For instance, we have a d-dimensional (or d-way) tensor $\mathcal{V} \in \mathbb{R}^{l_1\times l_2\times\cdots\times l_d}$, where entries are indexed by d indices $p_1, p_2, \ldots, p_k, \ldots, p_d$. Each p_k is within $[1, l_k]$ for $k = 1, 2, \ldots, d$. The high-order tensor $\mathcal{V}$ can then be decomposed and represented using a collection of d tensor cores $\mathcal{G}^{(k)} \in \mathbb{R}^{r_{k-1}\times l_k\times r_k}$ with $k \in [1, d]$. Each entry in $\mathcal{V}$ can be reconstructed by these tensor cores as:

$$\mathcal{V}_{p_1,p_2,\cdots,p_d} = \prod_{k=1}^{d} \mathbf{G}^{(k)}_{p_k}, \tag{1}$$

where $\mathbf{G}^{(k)}_{p_k} = \mathcal{G}^{(k)}(:, p_k, :) \in \mathbb{R}^{r_{k-1}\times r_k}$ is the p_k-th lateral slice of the k-th tensor core $\mathcal{G}^{(k)}$. In each tensor core, $r \in \{r_0, r_1, \cdots, r_d\}$ is called the tensor-train rank where both boundaries r_0 and r_d are fixed to 1. Due to the compact tensor representation scheme, the number of parameters to formulate a d-way tensor $\mathcal{V}$ is only $\sum_{k=1}^{d} r_{k-1} l_k r_k$ while the conventional explicit representation needs $\prod_{k=1}^{d} l_k$ parameters. We graphically show an example of tensor-train decomposition in Fig. 1 for better explanation. For a 2-way weight matrix, it has to be rearranged into a d-way tensor representation before conducting the tensor-train decomposition.

Tensor-Train Decomposition for Feed Forward Network Feed forward networks in the DETR model are sequentially connected linear layers, which take up most of the model size. We simplify (omit the bias) and denote the linear layers in FFN modules with the matrix-by-vector multiplication form as $y = \mathbf{W} \cdot f$, where input feature vector $f \in \mathbb{R}^N$, weight matrix $\mathbf{W} \in \mathbb{R}^{M\times N}$, and output feature vector

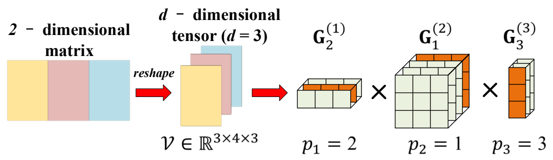


Figure 1: A 2-way weight matrix example for tensor-train decomposition.

$y \in \mathbb{R}^M$. Then we can rewrite this layer more explicitly with indices $y_i = \sum_{j=1}^{N} \mathbf{W}_{i,j} \cdot f_j$, where $i \in [1, M]$ and $j \in [1, N]$.

For the 2-way weight matrix $\mathbf{W}$ as well as the input feature f and output feature y in FFN modules, they should be rearranged to d-way tensor representations by the projection function ϕ as follows:

$$\begin{aligned}\phi_w &: \mathbb{R}^{M\times N} \to \mathbb{R}^{(m_1\times n_1)\times\cdots\times(m_d\times n_d)}, \phi_w(\mathbf{W}) = \mathcal{W},\\ \phi_y &: \mathbb{R}^{M} \to \mathbb{R}^{m_1\times m_2\times\cdots\times m_d}, \phi_y(y) = \mathcal{Y},\\ \phi_f &: \mathbb{R}^{N} \to \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}, \phi_f(f) = \mathcal{F}.\end{aligned} \tag{2}$$

We assume that both M and N can be factorized into two integer arrays with $M = \prod_{k=1}^{d} m_k$ and $N = \prod_{k=1}^{d} n_k$ respectively. Following the above equation (2), the expression of the tensorized weight can be changed from $\mathbf{W}_{i,j}$ to $\mathcal{W}_{i_1,j_1,i_2,j_2,\cdots,i_d,j_d}$ with $2d$-tuple indices.

Then in FFN modules, the 2-way weight $\mathbf{W}$ and the input feature f multiplication of linear layers can be represented in tensor format as follows:

$$\mathcal{Y}_{i_1,i_2,\cdots,i_d} = \sum_{j_1,j_2,\cdots,j_d} \mathcal{W}_{i_1,j_1,i_2,j_2,\cdots,i_d,j_d} \cdot \mathcal{F}_{j_1,j_2,\cdots,j_d}, \tag{3}$$

where $\mathcal{Y}$, $\mathcal{F}$, and $\mathcal{W}$ are the high-order tensor representations of y, f, and $\mathbf{W}$ as depicted in Eq. (2).

Following the preliminaries, the d-way weight tensor $\mathcal{W}$ can be further represented in the basic tensor-train form:

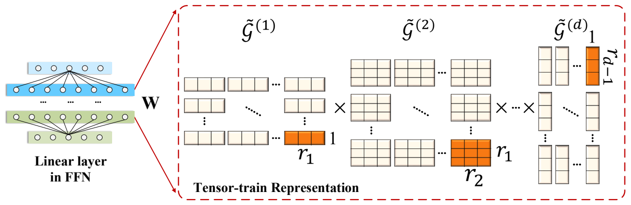
$$\mathcal{W}_{i_1,j_1,i_2,j_2,\cdots,i_d,j_d} = \prod_{k=1}^{d} \widetilde{\mathbf{G}}^{(k)}_{i_k,j_k}, \tag{4}$$

where $\widetilde{\mathbf{G}}^{(k)}_{i_k,j_k} = \widetilde{\mathcal{G}}^{(k)}(:, i_k, j_k, :)$ can be viewed as an $r_{k-1}\times r_k$ slice of the 4-d tensor core $\widetilde{\mathcal{G}}^{(k)} \in \mathbb{R}^{r_{k-1}\times m_k\times n_k\times r_k}$. Here the index l_k in $\mathcal{G}^{(k)} \in \mathbb{R}^{r_{k-1}\times l_k\times r_k}$ is supposed to be factorized as $l_k = m_k \cdot n_k$. We show the tensor-train decomposition of FFN weight matrix $\mathbf{W}$ in Fig. 2.

In our experiments, the tensor decomposition is applied to every $\mathbf{W}\cdot f$ operation in the FFN modules following Eq. (4) since linear layers are the most storage-intensive parts in the DETR models. The tensorized multiplication of the input feature and weights can then be represented as follows:

$$\mathcal{Y}_{i_1,i_2,\cdots,i_d} = \sum_{j_1,j_2,\cdots,j_d} \widetilde{\mathbf{G}}^{(1)}_{i_1,j_1} \widetilde{\mathbf{G}}^{(2)}_{i_2,j_2} \cdots \widetilde{\mathbf{G}}^{(d)}_{i_d,j_d} \cdot \mathcal{F}_{j_1,j_2,\cdots,j_d}. \tag{5}$$

In our tensorized DETR model, each FFN module consists of two tensor-compressed linear layers. The first linear layer has input length 256 and output length 2048 while



$\tilde{\mathcal{G}}^{(1)}$
$\tilde{\mathcal{G}}^{(2)}$
$\tilde{\mathcal{G}}^{(d)}$
W
Linear layer in FFN
r_1
r_2
r_{d-1}
Tensor-train Representation

Figure 2: Tensor-train decomposition of weight matrix $\mathbf{W}$ in the feed forward networks.

the second linear layer has the swapped input and output length. In the validation experiments, the vector length 256 (N) is factorized to $[2,4,4,4,2]$ and 2048 (M) is factorized to $[4,4,8,4,4]$. We have $d=5$. The tensor-train ranks are set to $\{1,4,4,4,4,1\}$.

Theoretical Benefits The main benefit of using tensorized linear is that the model size and computational complexity can be reduced. For a vanilla linear layer in the FFN module, the time complexity would be $\mathcal{O}(NM)$. Whereas in the tensorized linear layer, the final time complexity is $\mathcal{O}(d\hat{r}^2\hat{n}M)$, where $\hat{n}=\max_k(n_k)$, $\hat{r}$ is the maximal rank (ranks are equal in our experiments), and d is the dimensionality of a tensor. The space complexities are $\mathcal{O}(NM)$ and $\mathcal{O}(d\hat{r}^2\hat{m}\hat{n})$ respectively, where $\hat{m}=\max_k(m_k)$.

We can also derive the compression ratio ($\digamma$) in the FFN linear layers as the ratio between the number of weights in a vanilla layer and that in its tensor-compressed form:

$$\digamma = \frac{\prod_{k=1}^{d} n_k \cdot \prod_{k=1}^{d} m_k}{\sum_{k=1}^{d} r_{k-1} m_k n_k r_k}. \tag{6}$$

We give the numerical analysis with a simple example. Suppose we have a linear layer with input length 256 and output length 2048, we will need 524,288 parameters to represent the weight matrix. If we factorize the input vector into dimension $2\times4\times4\times4\times2$ and the output vector into dimension $4\times4\times8\times4\times4$, we only need 1088 parameters to get the weight matrix using a tensor-train rank 4; and if the tensor-train rank is 3, we will only need 624 parameters. As a result, compression ratios of the corresponding ranks are around 482× and 840× respectively.

The Gated Multi-head Attention Module

Formulation of the Learnable Gate Parameters Multi-head attention (MHA) is a powerful and ubiquitous module in the DETR model, which allows the network to obtain information from different mapping spaces. In vanilla DETR models, we usually have 8 heads in each attention module. However, not all the heads contribute equally for making predictions (Vig and Belinkov 2019). The redundant information causes the DETR model to overfit. We normalize the vanilla multi-head attention module by multiplying the single head representation $\mathbf{H}_i$ with a learnable gate parameter g_i. Consequently, the proposed GMHA can be written as:

$$\text{G-MultiHead}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Concat}\left(g_i \cdot \mathbf{H}_i\right)\mathbf{W}^O, \tag{7}$$

where $i\in\{1,\dots,n_h\}$ denotes the number of heads. Inspired by (Gal, Hron, and Kendall 2017; Maddison, Mnih, and Teh 2017), our gate parameters are independently learned from the binary hard concrete distributions. Given the learnable parameter set $\mathbf{q}$ for all heads in a MHA module, our gate parameter set $\mathbf{g}$ can be derived as follows:

$$\mathbf{s} = \text{Sigmoid}\left(\left(\mathbf{q}+\log u - \log\left(1-u\right)\right)/T\right), \tag{8}$$

$$\bar{\mathbf{s}} = \mathbf{s}\left(\lambda-\mu\right)+\mu, \tag{9}$$

$$\mathbf{g} = \text{clamp}\left(\bar{\mathbf{s}},0,1\right). \tag{10}$$

The set $\mathbf{q}$ only contains n_h parameters which is the number of heads. In the above Eq. (8), u is a random noise drawn from the uniform distribution $\mathcal{U}(0,1)$ and can push the $\mathbf{s}$ away from middle values. $T\in(0,1)$ is the temperature that controls the magnitude of value. As we can see from the Eq. (8), the range of $\mathbf{s}$ is within $(0,1)$. Since we want the heads can be totally opened or closed in GMHA modules, we introduce two parameters μ,λ that can stretch the valve of $\mathbf{s}$ to (μ,λ) with $\mu<0$ and $\lambda>1$ as described in Eq. (9). Then we can apply a hard-sigmoid function to clip the value of gate $\mathbf{g}$ into range $[0,1]$ following Eq. (10). In our experiments, the μ and λ are set to -0.1 and 1.1 respectively. T is selected as 0.33.

To evaluate the performance of the GMHA module, we conduct a series of experiments with GMHA applied to only encoder self-attention and all multi-head attention (i.e. encoder and decoder self-attention and cross-attention from encoder to decoder). Based on the experimental results, if not specified, the GMHA module is only applied to encoder self-attention.

Learning of the Gate Parameters Since some of the heads contain redundant information that hinders the model performance, we would like to down-weight or even disable these heads. Inspired by (Louizos, Welling, and Kingma 2017), we propose a penalty function during the training of tensorized DETR models that would push the models to switch off those redundant heads. The penalty loss function L_p can be expressed as follows:

$$L_p\left(g_1,g_2,\cdots,g_{n_h}\right)=\textstyle\sum_{i=1}^{n_h}\left(1-P\left(g_i=0\mid\varphi_i\right)\right), \tag{11}$$

where g_i is the gate parameter. P is the probability mass derived from the hard concrete distribution and φ_i is the distribution parameter. This penalty function is a relaxation of L_0 norm with the sum of probability mass of non-zero gates (Louizos, Welling, and Kingma 2017). During the training of our tensorized DETR models, the learning objective function $\mathcal{L}$ is a linear combination of the original DETR loss and the penalty loss function $\mathcal{L}=L_d+\rho\cdot L_p$, where ρ is a balancing parameter.

Low Bitwidth Compression

In the vanilla DETR models, the backbone network is another major source of the large model size. To obtain a fully compressed DETR model, we introduce the post-training quantization technique for low-bitwidth backbone network compression. This post-training quantization technique does not require any tedious retraining procedure and the full training dataset compared with QAT methods. Given the weight representation $\mathbf{W}$, we need to find an optimal quantization interval for constraining the floating-point values into a finite set of values. Following the uniform quantization formulation, we obtain the quantization point:

$$\mathbf{W}_q = \text{clamp}\left(\lfloor \frac{\mathbf{W}}{\Delta} \rceil, -2^{s-1}, 2^{s-1}-1\right), \tag{12}$$

where Δ is the quantization scale, s is the quantization level, and $\mathbf{W}_q$ is the quantized weight. $\lfloor \cdot \rceil$ denotes rounding to the nearest integer.

For post-training quantization, our goal is to learn an optimal mapping between the outputs of full-precision and low-bitwidth convolutions. This quantization procedure can be formulated as an optimization problem:

$$\underset{\Delta, \mathbf{W}_q}{\arg\min} \, \|y - \Delta \mathbf{W}_q \cdot f\|_F^2 , \tag{13}$$

where Δ is the scale to be learned, y is the full-precision convolution output, and f is the input feature. The similarity between the vanilla and quantized output is measured using the euclidean metric. This optimization problem for quantization can be solved in an iterative way. The quantization scale can be derived as:

$$\Delta = \frac{y^\top f^\top \mathbf{W}_q^\top}{\mathbf{W}_q f f^\top \mathbf{W}_q^\top}. \tag{14}$$

The quantization points can be optimized bit-by-bit following (Wang et al. 2020).

Experiments

Datasets

COCO (Lin et al. 2014) is used to validate our proposed method, which contains 118k training images and 5k validation images. Following the DETR baseline, we report bbox AP on the validation dataset for ablations and comparisons with state-of-the-art.

ImageNet-1k (Deng et al. 2009) is leveraged to calibrate and validate our quantization compressed backbone. The dataset consists of 1.28M training images and 50k validation images from total 1000 semantic categories.

Implementation Details

DETR Training We adopt the vanilla DETR as our baseline model which is implemented based on the `mmdetection`. The CNN backbone is the ImageNet pretrained ResNet-50 (He et al. 2016). All models are trained on 4 NVIDIA GTX 1080Ti GPUs with 2 images per GPU. We train the models by using AdamW optimizer with 150 epochs in total. The learning rates of transformer encoder-decoder and the CNN backbone are initialized to 5×10^{-5} and 5×10^{-6} respectively. The weight decay is set to 0.0001. The learning rates are divided by 10 at the decay step 100 epoch. The balancing parameter ρ for the penalty function is set to 0.1. As for the transformer implementations, we leverage 6 encoder layers and 6 decoder layers with embedded dimension 256. Each encoder and decoder layer has 8 attention heads. Following DETR, we leverage a simple data augmentation technique by resizing the input images with the short side ranging from 480 to 800 pixels and the long side at most 1333 pixels.

All the results of speed (FPS) in our experiments are measured under one NVIDIA GTX 1080Ti GPU. FPS is computed by averaging the speed for inferencing 2000 images from the COCO validation dataset.

Quantization ResNet-50 is employed as the basic backbone network of DETR models. We randomly sample 300 images from ImageNet-1k and COCO training datasets respectively as our calibration datasets.

Ablation Study

Effectiveness of the Tensorized Compression We perform ablation studies on the COCO dataset to verify the effectiveness of tensor decomposition. The experimental results are given in Table 1. We first reproduce the baseline model and it can achieve 40.1% AP as shown in Table 1. Then we show the results obtained by the proposed tensorized DETR (T-DETR) model. Note that the GMHA module and quantization are not leveraged in this experiment. As can be seen from Table 1, our proposed tensorized DETR model only has 2.5 points accuracy drop with 47.9MB transformer model size reduction. It should be noted that the vanilla ResNet-50 backbone has 90.0MB model size, which occupies most of the model size in one DETR model. Here we only consider the transformer architecture in the DETR model, and we achieve 3.3× model size reduction with the introduced tensor decomposition method.

Method	AP	Size	FPS	Mem
DETR	40.1	69.0 MB	16.6	7.9 GB
Ours	37.6	**21.1 MB**	15.6	7.5 GB

Table 1: Comparison between the vanilla DETR model and our tensorized DETR model. Size here is the transformer storage size. Mem denotes the training memory consumption.

In addition, as shown in Table 1, the tensorized DETR model has slightly lower FPS than the vanilla DETR model. Because tensor decomposition requires more mathematical operations instead of one highly optimized matrix multiplication during implementation (in PyTorch). However, these computation overheads are acceptable since we mainly aim at deeply compressing the model size. The experimental results can demonstrate that the proposed method benefits from the tensor decomposition.

Analysis About the Tensor-Train Rank The tensor-train decomposition rank r plays an important role in reducing the redundant parameters of the transformers. We provide a numerical analysis of different tensor-train ranks to show their impact on the model performance. The experiments are conducted based on the FFN modules with two linear layers in the DETR model. The input vector of length 256 (N) is factorized as $2 \times 4 \times 4 \times 4 \times 2$, and the output vector of length 2048 (M) is factorized as $4 \times 4 \times 8 \times 4 \times 4$. In most applications, ranks are manually selected and set to equal since we can not list all the rank values exhaustively. Here the ranks are selected from 3 to 5. The experimental results are summarized in Table 2. In this table, the ranks are fixed with a total of 6 numbers, for example, $\{1, 4, 4, 4, 4, 1\}$.

Method	Rank	#Param	$\digamma$	AP	FPS
Ours (T-DETR)	FFN	1,048,576	1×	40.1	16.6
	3	1,248	840×	36.9	15.9
	4	2,176	482×	37.6	15.6
	5	3,360	312×	37.7	15.6
Tucker (2009)	2	2,208	475×	37.4	9.0
Tucker (2009)	3	118,338	9×	37.7	8.8
Tucker-2 (2009)	4	14,283	73×	37.6	16.3

Table 2: Numerical analysis about the tensor-train rank and comparisons with other tensor decomposition methods. $\digamma$ denotes the compression ratio. AP and FPS are measured using corresponding DETR models.

As shown in Table 2, the tensor-train decomposition can achieve up to 840× parameter reduction with a slight accuracy drop. Moreover, we can see that the detection accuracy will slightly increase as the tensor-train rank grows; however, the number of parameters increases significantly. We have similar speed due to the powerful parallel computing capabilities of GPUs. As a result, to obtain the best accuracy and parameter number trade-off, we choose the tensor-train rank 4 as the default setting in our experiments.

We also make comparisons with other tensor compression methods in Table 2. When using ranks 2 and 3, tucker-compressed DETR models show lower compression ratio and similar detection accuracy. When the rank is 4, the parameters in one tucker-compressed FFN is 2,097,472, which are more than those in one vanilla FFN. The parameters in tucker representation will explode when the rank grows since they are proportional to r^{10}. Our method shows much better efficiency than the tucker decomposition method.

Effectiveness of the GMHA Module In this section, experiments are performed to demonstrate the effectiveness of the proposed GMHA module. The quantitative results are shown in Table 3. First, we integrate the proposed GMHA module into vanilla DETR models. It is shown in Table 3 that the GMHA module helps the vanilla DETR model achieve 1.6 points AP improvement. Furthermore, we can see that the tensorized DETR (T-DETR) model obtains 1.9 points accuracy improvement with the GMHA module. The experimental results can demonstrate the advantages of the GMHA module for suppressing the uninformative attention values and alleviating the overfitting problem.

We find that our GMHA module is more effective when applied to only encoder attentions. The experimental results are summarized in Table 4. In this experiment, GMHA is leveraged based on the tensor-compressed DETR model. As shown in Table 4, the obtained accuracy is 39.2% for models with GMHA module in all attentions, whereas the accuracy is 39.5% for models with GMHA module in only encoder attentions. Note that we only need 48 additional parameters

Method	AP	Size	FPS	Mem
DETR	40.1	69.0 MB	16.6	7.9 GB
+ GMHA	**41.7 (1.6 ↑)**	69.0 MB	16.3	7.9 GB
Ours (T-DETR)	37.6	21.1 MB	15.6	7.5 GB
+ GMHA	**39.5 (1.9 ↑)**	21.1 MB	15.2	7.5 GB

Table 3: Effect of the proposed GMHA module on model performance. Size is the transformer storage size.

for leveraging the GMHA modules into only encoder layers and 144 additional parameters for leveraging them into all encoder and decoder layers. Results came out that these gate parameters do not cause significant drop in the detection quality, and also do not increase the model size.

Attentions	AP	#Param $\mathbf{g}$	Size	FPS
Only encoder	39.5	48	21.1 MB	15.2
All heads	39.2	144	21.1 MB	14.9

Table 4: Detection performance for GMHA modules in different attentions. #Param represents the additional gate parameters. Size is the transformer storage size.

Furthermore, we show an example of the learned gate parameters $\mathbf{g}$ in Fig. 3. The first observation is that most of the gates are totally opened or closed, which means the gate values are either 1 or 0. The second observation is that the first head contains the most redundant information since all the first heads have gate parameters 0. We hypothesize this is because edge pixels of images or features contribute little to the detection results. The third observation is that the first two layers are less important than the following layers. The first two layers have three closed gates while the other layers mostly have one closed gate. We argue that deep feature representations are more informative than the shallow features in the transformer architecture.

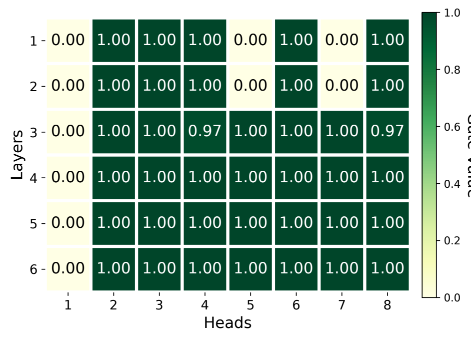


Figure 3: Heap map of the learned gate parameters from the encoder layers.

Quantization Evaluation For ImageNet-1k evaluation, the PyTorch pretrained ResNet-50 is leveraged. We quantize all the weight layers into 4-bit and 8-bit while keeping the activations in full precision (FP32). The results are shown in

Method	Backbone	Size	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
RetinaNet (Lin et al. 2017)	ResNet-50	145.1 MB	36.5	55.4	39.1	20.4	40.3	48.1
Faster-RCNN DC5 (Ren et al. 2015)	ResNet-50	631.8 MB	37.2	58.3	39.9	19.5	41.4	50.4
Faster-RCNN FPN (Ren et al. 2015)	ResNet-50	159.5 MB	37.4	58.1	40.4	21.2	41.0	48.1
Mask-RCNN (He et al. 2017)	ResNet-50	169.6 MB	38.2	58.8	41.1	21.9	40.9	49.5
Faster-RCNN FPN (Ren et al. 2015)	ResNet-101	232.2 MB	39.4	60.1	43.1	22.4	43.7	51.1
FCOS-GN (Tian et al. 2019)	ResNet-50	123.5 MB	36.6	56.0	38.8	21.0	40.6	47.0
RepPoints (Yang et al. 2019)	ResNet-50	140.8 MB	37.0	56.7	39.7	20.4	41.0	49.0
Deformable-DETR† (Zhu et al. 2020)	ResNet-50	156.0 MB	39.7	60.1	42.4	21.2	44.3	56.0
YOLOS-S (Fang et al. 2021)	DeiT-S	351.3 MB	36.1	56.4	37.1	15.3	38.5	56.2
UP-DETR (Dai et al. 2021)	ResNet-50	164.0 MB	40.5	60.8	42.8	19.0	44.4	60.0
Baseline (DETR)	ResNet-50	159.0 MB	40.1	60.6	42.0	18.3	43.3	59.5
Ours (T-DETR)	ResNet-50	111.1 MB	39.5	59.9	41.6	18.7	42.5	58.0
Ours (T-DETR)	R-50 8-bit	**43.6 MB**	39.5	59.8	41.6	18.8	42.4	58.0
Ours (T-DETR)	R-50 4-bit	**33.4 MB**	37.9	57.9	39.8	17.3	40.6	56.3

Table 5: The quantitative comparisons with state-of-the-art methods on the COCO validation dataset. For fair comparison, the results are measured under `mmdetection` implementation. Size is the full model storage size here. † denotes single-scale Deformable-DETR. ResNet-101 is not desirable in our method since we aim to reduce the model size.

Table 6. As shown in this table, the 8-bit quantized model has higher accuracy than the full-precision model. As for the 4-bit quantized model, there is only 0.64 / 0.27 accuracy drop from the full-precision model. In the classification evaluation, the size of the full-precision model is 97.7MB while the sizes of 8-bit and 4-bit quantized models would be 24.5MB and 12.3MB. Note that there are no linear layers in the backbone of DETR models, thus the backbone model size would be 90.0MB.

FP32	**8-bit**	**4-bit**
76.15 / 92.87	**76.18 / 92.95**	**75.51 / 92.60**

Table 6: Performance comparison between full-precision and quantized backbone networks. Acc (%): top-1 / top-5.

Furthermore, we quantize the backbone network in DETR models and measure the performance on the COCO validation dataset. The results are summarized in Table 7. The results in this table show that the 8-bit quantized backbones lead to no accuracy degradation. For vanilla DETR and our tensor-compressed DETR models, the bbox AP results are the same for both full-precision and 8-bit quantized backbones. We observe a significant accuracy drop when quantization is extended to the tensorized transformers. The experimental results demonstrate that quantization is an effective way for our backbone compression in DETR models.

Comparison with State-of-the-arts

Research works focusing on compressing detection transformers are limited; therefore in this section, we directly provide the quantitative comparisons with other object detection methods on the COCO validation dataset. To make the comparisons fair, we report the results based on the `mmdetection` implementation rather than from the original paper. The results are summarized in Table 5. First and foremost, our tensorized DETR model with an 8-bit

Backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
R-50	40.1	60.6	42.0	18.3	43.3	59.5
R-50 8-bit	40.1	60.7	42.0	18.5	43.4	59.6
R-50	39.5	59.9	41.6	18.7	42.5	58.0
R-50 8-bit	39.5	59.8	41.6	18.8	42.4	58.0
R-50 4-bit	37.9	57.9	39.8	17.3	40.6	56.3

Table 7: Quantitative comparisons between full-precision and quantized DETR backbones. The top and bottom rows show the results of the baseline DETR and our T-DETR respectively.

quantized backbone achieves 3.7× model size compression with only 0.6 points accuracy drop. Moreover, with a 4-bit backbone, we obtain 4.8× model size compression with 2.2 points accuracy drop. The experimental results show that our compressed models can maintain competitive detection accuracy against state-of-the-art methods.

Conclusion

In this paper, we demonstrate that tensor decomposition is a promising way to efficiently compress the detection transformer. By incorporating the tensor-train decomposition into FFN modules, we obtain up to 840× parameter reduction under comparable detection accuracy. A novel gated multi-head attention module with limited parameters is proposed for down-weighting the redundant attention values. Based on the GMHA module, the significant detection accuracy drop can be circumvented. With further low-bitwidth quantization techniques, we achieve up to 4.8× whole DETR model compression. Extensive experiments are performed on the COCO benchmark to validate our proposed methods. The results show we can achieve 39.5% AP after compression which is only 0.6 percentage points lower than the vanilla DETR model.

Acknowledgements

This work is supported by the National Key Research and Development Program of China under grant 2019YFB2205005.

References

Cai, Z.; and Vasconcelos, N. 2018. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 6154–6162.

Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; and Zagoruyko, S. 2020. End-to-end object detection with transformers. In *Proceedings of the European Conference on Computer Vision*, 213–229.

Chen, K.; Wang, J.; Pang, J.; Cao, Y.; Xiong, Y.; Li, X.; Sun, S.; Feng, W.; Liu, Z.; Xu, J.; Zhang, Z.; Cheng, D.; Zhu, C.; Cheng, T.; Zhao, Q.; Li, B.; Lu, X.; Zhu, R.; Wu, Y.; Dai, J.; Wang, J.; Shi, J.; Ouyang, W.; Loy, C. C.; and Lin, D. 2019. MMDetection: Open MMLab Detection Toolbox and Benchmark. *arXiv preprint arXiv:1906.07155*.

Dai, Z.; Cai, B.; Lin, Y.; and Chen, J. 2021. Up-detr: Unsupervised pre-training for object detection with transformers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1601–1610.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 248–255.

Fang, Y.; Liao, B.; Wang, X.; Fang, J.; Qi, J.; Wu, R.; Niu, J.; and Liu, W. 2021. You Only Look at One Sequence: Rethinking Transformer in Vision through Object Detection. *arXiv preprint arXiv:2106.00666*.

Gal, Y.; Hron, J.; and Kendall, A. 2017. Concrete dropout. *arXiv preprint arXiv:1705.07832*.

Gusak, J.; Kholiavchenko, M.; Ponomarev, E.; Markeeva, L.; Blagoveschensky, P.; Cichocki, A.; and Oseledets, I. 2019. Automated multi-stage compression of neural networks. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*.

Han, S.; Mao, H.; and Dally, W. J. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

He, K.; Gkioxari, G.; Dollár, P.; and Girshick, R. 2017. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, 2961–2969.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 1389–1397.

Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2016. Binarized neural networks. In *Advances in Neural Information Processing Systems*, volume 29.

Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; and Kalenichenko, D. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2704–2713.

Khoram, S.; and Li, J. 2018. Adaptive quantization of neural networks. In *International Conference on Learning Representations*.

Kolda, T. G.; and Bader, B. W. 2009. Tensor Decompositions and Applications. *SIAM Review*, 51(3): 455–500.

Kuchaiev, O.; and Ginsburg, B. 2017. Factorization tricks for LSTM networks. *arXiv preprint arXiv:1703.10722*.

Law, H.; and Deng, J. 2018. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision*, 734–750.

Li, D.; Wang, X.; and Kong, D. 2018. Deeprebirth: Accelerating deep neural network execution on mobile devices. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; and Dollár, P. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, 2980–2988.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft coco: Common objects in context. In *Proceedings of the European Conference on Computer Vision*, 740–755.

Louizos, C.; Welling, M.; and Kingma, D. P. 2017. Learning sparse neural networks through L_0 regularization. *arXiv preprint arXiv:1712.01312*.

Maddison, C.; Mnih, A.; and Teh, Y. 2017. The concrete distribution: A continuous relaxation of discrete random variables. In *Proceedings of the International Conference on Learning Representations*.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proceedings of the European Conference on Computer Vision*, 525–542.

Redmon, J.; and Farhadi, A. 2017. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7263–7271.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, 91–99.

Sun, Z.; Cao, S.; Yang, Y.; and Kitani, K. M. 2021. Rethinking transformer-based set prediction for object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, 3611–3620.

Tian, Z.; Shen, C.; Chen, H.; and He, T. 2019. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 9627–9636.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, 5998–6008.

Vig, J.; and Belinkov, Y. 2019. Analyzing the Structure of Attention in a Transformer Language Model. In *Proceedings of the ACL Workshop BlackboxNLP*, 63–76.

Wang, K.; Liu, Z.; Lin, Y.; Lin, J.; and Han, S. 2019. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8612–8620.

Wang, P.; Chen, Q.; He, X.; and Cheng, J. 2020. Towards accurate post-training network quantization via bit-split and stitching. In *International Conference on Machine Learning*, 9847–9856.

Wang, W.; Sun, Y.; Eriksson, B.; Wang, W.; and Aggarwal, V. 2018. Wide compression: Tensor ring nets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 9329–9338.

Yang, Z.; Liu, S.; Hu, H.; Wang, L.; and Lin, S. 2019. Reppoints: Point set representation for object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, 9657–9666.

Yin, M.; Liao, S.; Liu, X.-Y.; Wang, X.; and Yuan, B. 2021. Towards Extremely Compact RNNs for Video Recognition With Fully Decomposed Hierarchical Tucker Structure. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 12085–12094.

Zhang, S.; Chi, C.; Yao, Y.; Lei, Z.; and Li, S. Z. 2020. Bridging the gap between anchor-based and anchor-free detection via adaptive training sample selection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 9759–9768.

Zheng, M.; Gao, P.; Wang, X.; Li, H.; and Dong, H. 2020. End-to-end object detection with adaptive clustering transformer. *arXiv preprint arXiv:2011.09315*.

Zhou, A.; Yao, A.; Guo, Y.; Xu, L.; and Chen, Y. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*.

Zhou, X.; Wang, D.; and Krähenbühl, P. 2019. Objects as points. *arXiv preprint arXiv:1904.07850*.

Zhu, X.; Su, W.; Lu, L.; Li, B.; Wang, X.; and Dai, J. 2020. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*.

Zhuang, Z.; Tan, M.; Zhuang, B.; Liu, J.; Guo, Y.; Wu, Q.; Huang, J.; and Zhu, J. 2018. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, 883–894.