DOI: 10.19884/j.1672-5220.202403010

Greedy Pruning Algorithm for DETR Architecture Networks Based on Global Optimization

HUANG Qiubo*, XU Jingsai, ZHANG Yakui, WANG Mei, CHEN Dehua
School of Computer Science and Technology, Donghua University, Shanghai 201620, China

Abstract: End-to-end object detection Transformer (DETR) successfully established the paradigm of the Transformer architecture in the field of object detection. Its end-to-end detection process and the idea of set prediction have become one of the hottest network architectures in recent years. There has been an abundance of work improving upon DETR. However, DETR and its variants require a substantial amount of memory resources and computational costs, and the vast number of parameters in these networks is unfavorable for model deployment. To address this issue, a greedy pruning (GP) algorithm is proposed, applied to a variant denoising-DETR (DN-DETR), which can eliminate redundant parameters in the Transformer architecture of DN-DETR. Considering the different roles of the multi-head attention (MHA) module and the feed-forward network (FFN) module in the Transformer architecture, a modular greedy pruning (MGP) algorithm is proposed. This algorithm separates the two modules and applies their respective optimal strategies and parameters. The effectiveness of the proposed algorithm is validated on the COCO 2017 dataset. The model obtained through the MGP algorithm reduces the parameters by 49% and the number of floating point operations (FLOPs) by 44% compared to the Transformer architecture of DN-DETR. At the same time, the mean average precision (mAP) of the model increases from 44.1% to 45.3%.

Keywords: model pruning; object detection Transformer (DETR); Transformer architecture; object detection
CLC number: TP391.4; TP301.6 **Document code**: A
Article ID: 1672-5220(2025)01-0096-10

Open Science Identity (OSID)

0 Introduction

End-to-end object detection Transformer (DETR)[1] is a milestone in the application of the Transformer architecture in the field of object detection. It features an encoder-decoder main architecture that integrates the attention mechanism and cleverly uses the Hungarian algorithm to solve the optimal matching problem between predicted bounding boxes and actual ground truth (GT) boxes. Not only does it perform well in the field of object detection, surpassing most traditional convolutional neural networks (CNNs), but it also ranks among the top in downstream tasks such as action recognition and interaction detection. However, DETR also has several drawbacks. Firstly, its convergence speed is slow, requiring up to 500 epochs to achieve convergence. Secondly, its detection performance for small targets is not satisfactory. Thirdly, the DETR architecture has a large number of parameters, which means that its deployment requires high capacity and memory, making it difficult to apply to lightweight devices.

In response to the drawbacks of DETR, many variant models have been developed to improve it. Deformable DETR[2] enhances the performance of small object detection by combining deformable convolutions with the original DETR structure. Collaborative-DETR (Co-DETR)[3] proposes a novel co-training and mixed assignment training scheme, which learns a more efficient detector from a variety of label assignment methods and significantly accelerates convergence. Dynamic anchor boxes-DETR (DAB-DETR)[4] reintroduces anchor boxes into DETR, where this modeling approach allows the query to act as a positional prior, thus improving performance. Denoising-DETR (DN-DETR)[5] builds upon DAB-DETR and introduces the concept of denoising. By employing tricks such as adding noise to the network's labels and true boxes, the convergence of DETR is greatly accelerated. While these improvements have enhanced the accuracy of DETR and sped up its convergence, they have also introduced many new modules that add to the original load of DETR, thus increasing the overall parameters and the computational costs of the model.

Model pruning is an important method in the field of model compression to reduce the parameters in a model. This compression technique identifies and eliminates redundant parameters in heavyweight models through various methods, thereby enabling the light weight of the model for deployment on lightweight devices. At the same time, the accuracy loss of the model due to this

Received date: 2024-03-22
Foundation item: Shanghai Municipal Commission of Economy and Information Technology, China (No. 202301054)
* Correspondence should be addressed to HUANG Qiubo, email: huangturbo@dhu.edu.cn
Citation: HUANG Q B, XU J S, ZHANG Y K, et al. Greedy pruning algorithm for DETR architecture networks based on global optimization[J]. *Journal of Donghua University (English Edition)*, 2025, 42(1): 96-105.

中国知网 https://www.cnki.net

process is negligible. It is believed that the DETR architecture contains a considerable number of redundant parameters, which could decrease the model's accuracy to some extent. In response to the aforementioned issue, inspired by the pruning methods used in traditional CNNs as referenced in Ref. [6], a greedy pruning (GP) algorithm is proposed. This algorithm is specifically designed for pruning the Transformer module's encoder and decoder components in the DETR architecture. Utilizing a greedy strategy, the algorithm iteratively retains neurons in the layers designated for pruning and eliminates the remaining redundant neurons based on the global changes in the network. Further, considering the distinct functions of the multi-head attention (MHA) and feed-forward network (FFN) modules in the Transformer architecture, a modular GP (MGP) algorithm is proposed. This algorithm first applies GP to all MHA modules and then proceeds to prune all FFN modules.

Theoretically, two methods are applicable to DETR architecture networks. Given the extremely slow convergence of the vanilla DETR, the experiment was validated on DN-DETR, which is one of the best current improvements on DETR. The pruning of this network is illustrated in Fig. 1, where we apply the proposed pruning method to the encoder and decoder components of the Transformer module within DN-DETR.

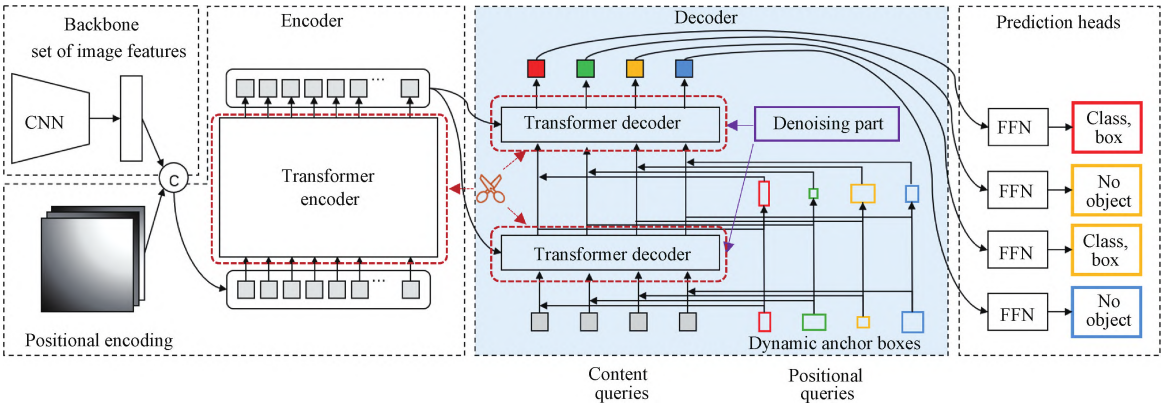

Fig. 1 Pruning of encoder and decoder components of the Transformer module within DN-DETR

In summary, the contributions of this research are as follows.

1) In response to the redundant parameters in the DETR architecture, a GP algorithm based on global optimization is proposed. This algorithm specifically prunes the encoder and decoder components of the Transformer module within the DETR architecture.

2) Recognizing the distinct functions of the MHA and FFN within the Transformer module, an MGP algorithm is proposed, separately pruning the MHA and FFN components of the Transformer.

3) Experiments are conducted on the classic object detection dataset COCO 2017, using the state-of-the-art variants of DETR and DN-DETR[5], to validate the effectiveness of both algorithms proposed.

1 Related Work

1.1 DETR and its variants

The introduction of DETR by Carion et al.[1] has made a significant impact in the field of object detection based on the Transformer architecture. DETR views object detection as a direct set prediction problem and employs a Transformer encoder-decoder architecture that enforces unique predictions through bipartite graph matching. Its innovative use of learned object queries allows for direct output of final predictions. Experiments on the COCO dataset have demonstrated its superior accuracy. However, its slow convergence, poor performance on small objects and high computational costs have been criticized. To address these issues, a substantial number of state-of-the-art improvements to DETR have been developed.

Zhu et al.[2] proposed an improved network called Deformable DETR. It featured a deformable attention module that focused on a small set of sampling locations, acting as a pre-filter to highlight significant features from the entire feature map. Roh et al.[7] addressed the issue of excessive encoder tokens and the additional computational costs in Deformable DETR. They then proposed Sparse DETR, which selectively updated only the tokens expected to be referenced by the decoder. In addition, an auxiliary detection loss was applied to the selected tokens in the encoder, which improved the performance while minimizing computational overhead. Meng et al.[3] tackled the Conditional DETR by developing a mechanism to generate conditional spatial queries from the decoder embedding, which were then utilized for the decoder's multi-head cross-attention process. This narrowed down the spatial range for localizing the distinct regions for object classification and box regression, thus relaxing the dependence on the content embeddings and easing the training. Liu et al.[4] presented a novel query formulation using dynamic anchor boxes for DETR. This new formulation directly used box coordinates as queries in Transformer decoders and dynamically updated them layer-by-layer. Using box coordinates not only helped to use explicit positional

中国知网 https://www.cnki.net

priors to improve the query-to-feature similarity and eliminate the slow training convergence issue in DETR but also modulated the positional attention map using the box width and height information.

Li et al.[5] suggested that the main reason for the slow convergence of the DETR architecture network was the discrete nature of the bipartite graph matching and the randomness in model training. This led to the matching of GT becoming a dynamic and unstable process. To address this problem, they proposed a novel denoising training method to speed up DETR training. Specifically, they additionally fed GT bounding boxes with noises into the Transformer decoder and trained the model to reconstruct the original boxes, which effectively reduced the bipartite graph matching difficulty and led to faster convergence. More importantly, the method was universal and could be easily plugged into any DETR-like method by adding dozens of lines of code to achieve a remarkable improvement. Experimental results showed that in just 12 epochs, this method could surpass most state-of-the-art methods. Given the encouraging effects of DN-DETR in accelerating DETR, pruning research of this paper on the DETR architecture network chooses this network as the benchmark for study.

1.2 Model pruning

Model pruning is an effective method for reducing the model parameters and accelerating model inference. The earliest recognized work on pruning neural networks was proposed by Han et al.[8]. They defined the pruning process of redundant connections using a three-step approach. This approach, both standardized and effective, established the paradigm for pruning and has become the benchmark reference for subsequent pruning workflow definitions. The pruning work proposed by Liu et al.[9] was referred to as network slimming. This was achieved by enforcing channel-level sparsity in the network in a simple but effective way. Different from many existing approaches, the proposed approach directly applied to modern CNNs, introduced minimum overhead to the training process, and required no special software/hardware accelerators for the resulting models. Ye et al.[6] proposed an alternative approach by introducing a GP algorithm for CNNs. The proposed algorithm guaranteed that the discrepancy between the pruned network and the original network decays at an exponentially fast rate with respect to the size of the pruned network, under weak assumptions that apply to most practical settings. All the aforementioned researches focus on pruning research for traditional CNNs. These researches effectively reduce the memory requirements of the models, accelerate the inference speed, and maintain the accuracy of the models before and after pruning.

In recent years, the Transformer architecture, based on the attention mechanism, has gained immense popularity. Initially introduced in the field of natural language processing (NLP), its application in computer vision has been rapidly expanding. Vision Transformer (ViT) proposed by Dosovitskiy et al.[10] is a milestone in the application of the Transformer architecture to visual tasks. This work ignited the research in the visual domain of Transformers. Like traditional CNNs, the Transformer architecture also suffers from issues such as parameter redundancy, high computational costs, and slow inference. As a result, a lot of work on applying pruning techniques to the Transformer architecture has emerged. The research by Zhu et al.[11] is one of the earlier studies focusing on pruning the Transformer architecture. They presented a ViT pruning approach, which identified the impacts of dimensions in each layer of the Transformer and then executed pruning accordingly. By encouraging dimension-wise sparsity in the Transformer, important dimensions automatically emerged. A large number of dimensions with low importance scores could be discarded to achieve a high pruning ratio without significantly compromising accuracy. There is also some work that focuses on pruning the aforementioned DETR architecture network. Sun et al.[12] proposed the optimization of the DETR using structured pruning through sparsity-induced pruning, aiming to improve its inference speed and reduce computational costs. They adjusted the importance of module outputs through parameter scaling factors and sparse regularization terms and optimized the parameter scaling factors using an improved accelerated proximal gradient (APG) method. Most of the research on pruning the Transformer architecture follows the established workflow definitions of neural networks, while also introducing innovative techniques, and making the pruning of the Transformer architecture both simple and effective.

We propose a GP algorithm for the DN-DETR network that adapts the traditional CNN pruning algorithm from Ref. [6] to the Transformer architecture, innovatively proposing our Transformer architecture pruning algorithm. This adaptation represents a significant advancement in the field, applying established techniques to the newer, more complex Transformer architecture.

2 Methods

2.1 Overall pruning process

Given the DETR-like network, its structure comprises several key components. The backbone network is responsible for extracting features from images, the encoder component encodes these extracted features, the decoder component decodes the encoded features, and the detection head generates specific downstream outputs. Both the encoder and decoder components contain the MHA module and FFN module. It can be formally represented as

$$D(a) = B \circ T_{A1} \circ T_{F1} \circ T_{A2} \circ T_{F2} \circ \cdots \circ T_{Al} \circ T_{Fl} \circ L_{cls}, \quad (1)$$

where D denotes the DN-DETR; a is an input image; B denotes the backbone network; T_{Ai}, $i \in [1, l]$ denotes the MHA module; T_{Fi}, $i \in [1, l]$ denotes the FFN module; L_{cls} denotes the detection head module. The

中国知网 https://www.cnki.net

operation ∘ denotes the connection between the preceding and following modules. Pruning each T_{Al} and T_{Fl} to t_{Al} and t_{Fl}, respectively, the pruned network $D_p(a)$ can be represented as

$$D_p(a) = B \circ t_{A1} \circ t_{F1} \circ t_{A2} \circ t_{F2} \circ \cdots \circ t_{Al} \circ t_{Fl} \circ L_{cls}. \quad (2)$$

Figure 2 illustrates the schematic of pruning Transformer architecture of the DN-DETR network. Add denotes residual connections; Norm denotes normalization layers; MLP denotes multi-layer perceptron; $\boldsymbol{V}$, $\boldsymbol{K}$ and $\boldsymbol{Q}$ stand for three vectors of value, key and query, respectively; M and N represent the number of stacks; (x, y, w, h) denotes the center point (x, y), width w and height h of the anchor boxes; $(\Delta x, \Delta y, \Delta w, \Delta h)$ denotes the calculated offsets; (x', y', w', h') denotes the parameters after regression; (w_{ref}, h_{ref}) denotes the intermediate values of the calculation process.

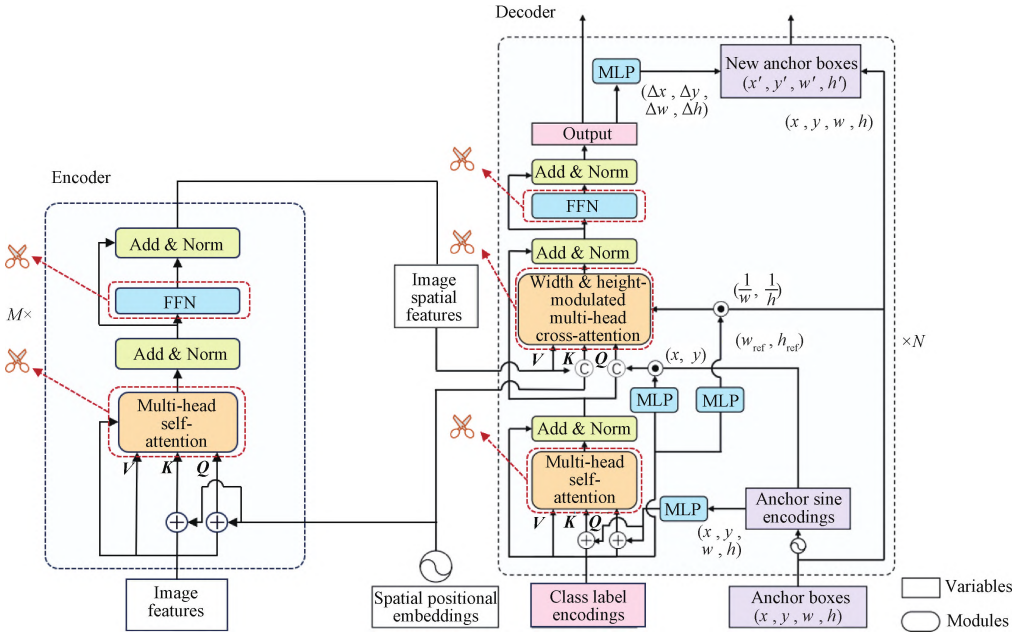

Fig. 2 Illustration of pruning Transformer architecture of DN-DETR network

After pruning to create the new model, the original model parameters are correctly transplanted into the new model. Finally, fine-tuning is performed to complete the process.

2.2 Pruning of modules

2.2.1 Pruning of linear layer

Given the lth linear layer L_l in the DN-DETR network, it can be represented as

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \quad (3)$$

where $\boldsymbol{x}$ is the input feature of the linear layer, and $\boldsymbol{x} \in \mathbb{R}^s$; $\boldsymbol{W}$ is the weight matrix, and $\boldsymbol{W} \in \mathbb{R}^{s \times t}$; $\boldsymbol{b}$ is the bias vector, and $\boldsymbol{b} \in \mathbb{R}^t$; $\boldsymbol{y}$ is the output feature of the linear layer, and $\boldsymbol{y} \in \mathbb{R}^t$. Following this layer, we define a channel selection layer L_{cs}:

$$\boldsymbol{z} = \boldsymbol{V}_{cs} \odot \boldsymbol{y}. \quad (4)$$

In the L_{cs} layer, the parameter $\boldsymbol{V}_{cs}$ is a one-dimensional vector, and $\boldsymbol{V}_{cs} \in \mathbb{R}^t$; $\boldsymbol{z}$ is the output feature of the channel selection layer, and $\boldsymbol{z} \in \mathbb{R}^t$. The operation $\odot$ denotes element-wise multiplication. This layer is a one-dimensional vector that corresponds one-to-one with the output of the linear layer L_l, and it is used to assess the performance of neurons in the L_l layer. Initially, all elements of $\boldsymbol{V}_{cs}$ are set to 1, meaning they do not alter the original forward output. Through iterative updates of $\boldsymbol{V}_{cs}$'s parameters, the final values of the elements in $\boldsymbol{V}_{cs}$ will range between 0 and 1. Ultimately, neurons corresponding to the components in $\boldsymbol{V}_{cs}$ that are greater than 0 are retained. This process is illustrated in Fig. 3. v_1, v_2, v_3, v_4 and v_5 denote the parameters of $\boldsymbol{V}_{cs}$.

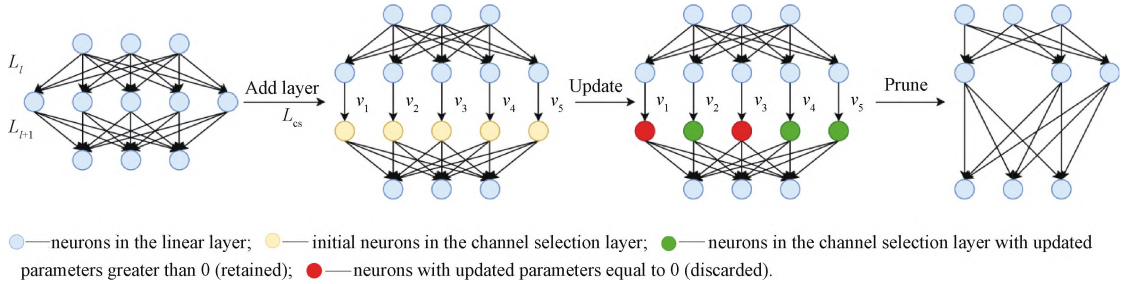

Fig. 3 Pruning of linear layer

中国知网 https://www.cnki.net

Algorithm 1 describes the iterative update algorithm for the parameters of the channel selection layer based on the global network.

Firstly, define the linear layer L_l to be pruned and the corresponding channel selection layer L_{cs}, and define the vector parameters for L_{cs}:

$$\boldsymbol{V}_{cs} = [v_1, v_2, \cdots, v_t] \in \mathbb{R}^t,\ v_i = 1,\ i \in [1, t], \quad (5)$$

where v_i denotes the parameters of the layer; t is the number of parameters in the layer. For each layer, before beginning the pruning iteration, set $v_i = 0$. Define the regression difference loss between the network with the pruned layer and the original network as $\mathbb{D}$, and define the candidate set C:

$$C = \{v_i \text{ in } \boldsymbol{V}_{cs} \mid v_i = 0\}. \quad (6)$$

During the process, only a small subset of the dataset D_{set} is used for inference. In the greedy selection iteration, each time one parameter v_i from the candidate set C is set to 1, and then the model is inferred. This cycle continues until each parameter in C has been selected once, resulting in N_c different values of $\mathbb{D}$, where N_c is the number of elements in C. The top e parameters v_i that minimize $\mathbb{D}$ are then selected, $i \in \{c_1, c_2, \cdots, c_e\}$. Following the iterative optimization approach for convolutional layers described in Ref. [6], we define the coefficient γ_k:

$$\gamma_k = 1/(k_e + 1), \quad (7)$$

where k denotes the kth iteration. The parameters in $\boldsymbol{V}_{cs}$ are then updated as

$$\boldsymbol{V}_{cs} = (1 - \gamma_k)\boldsymbol{V}_{cs}, \quad (8)$$

$$v_i = \gamma_k,\ i \in \{c_1, c_2, \cdots, c_e\}. \quad (9)$$

The iteration stops when $\mathbb{D}$ falls below a predefined threshold ε. At the same time, the parameters in L_{cs} are considered fully updated. In each iteration, the algorithm selects the best-performing m parameters, similar to a process of greedy selection. Therefore, it is called the "GP algorithm".

Algorithm 1: iterative update of parameters for the channel selection layer

Input: the linear layer L_l with index l for pruning;
the channel selection layer L_{cs} for L_l;
the number of selections e;
the convergence criterion ε;
the mini-batch dataset D_{set}.

Output: the updated channel selection layer L_{cs}.

1: **Initialization**: the parameter vector for L_{cs}: $\boldsymbol{V}_{cs} = [v_1, v_2, \cdots, v_t] \in \mathbb{R}^t$, $v_i = 1$, $i \in [1, t]$;
candidate set: $C = \{v_i \text{ in } \boldsymbol{V}_{cs} \mid v_i = 0\}$;
regression discrepancy loss between $D(a)$ and $D_p(a)$: $\mathbb{D}$;
Iteration round: $k = 0$.
2: **while** $\mathbb{D} >= \varepsilon$ **do**
3: Obtain the top e elements v_i from C that minimize $\mathbb{D}$ under D_{set}, $i \in \{c_1, c_2, \cdots, c_e\}$.
4: $\gamma_k = 1/(k_e + 1)$.
5: $\boldsymbol{V}_{cs} = (1 - \gamma_k)\boldsymbol{V}_{cs}$.
6: $v_i = \gamma_k$, $i \in \{c_1, c_2, \cdots, c_e\}$.
7: k += 1.
8: Update $\mathbb{D}$.
9: **end while**

This algorithm, while drawing inspiration from the implementation in CNNs as seen in Ref. [6], introduces the following improvements.

1) The algorithm targets linear layers, as opposed to convolutional layers.

2) In Ref. [6], a Taylor estimation optimization method is used for convolutional layers. We improve upon the γ_k aspect of this (shown in Eq. 7), and the parameter updating method is revised to update $\boldsymbol{V}_{cs}$ (shown in Eqs. 8 and 9).

3) In the greedy iteration, m parameters are selected at each step instead of just 1, significantly speeding up the search process.

4) The overall algorithm is simplified, making it more understandable and user-friendly.

2.2.2 Pruning of MHA module

The attention mechanism operations in the Transformer architecture can be represented as

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}}{\sqrt{d}}\right)\boldsymbol{V}, \quad (10)$$

where $\boldsymbol{Q} \in \mathbb{R}^{n \times d}$; $K \in \mathbb{R}^{n \times d}$; $\boldsymbol{V} \in \mathbb{R}^{n \times d}$; n is the number of patches; d denotes the embedding dimension.

The MHA module can be represented as

$$\boldsymbol{Y} = \boldsymbol{X} + L_{out}(\text{Attention}(L_q(\boldsymbol{X}), L_k(\boldsymbol{X}), L_v(\boldsymbol{X}))), \quad (11)$$

where $\boldsymbol{Y}$ is the output feature of the MHA module, and

中国知网 https://www.cnki.net

$\boldsymbol{Y} \in \mathbb{R}^{n\times d}$; $\boldsymbol{X}$ is the input feature of the MHA module, and $\boldsymbol{X} \in \mathbb{R}^{n\times d}$; L_{q}, L_{k} and L_{v} are linear layers responsible for mapping $\boldsymbol{X}$ to $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$ respectively; L_{out} is an output linear layer. Following our pruning scheme, the pruned representation of this module can be expressed as

$$\boldsymbol{Y} = \boldsymbol{X} + L_{\mathrm{out}}^{*}(\mathrm{Attention}(L_{\mathrm{q}}^{*}(\boldsymbol{X}), L_{\mathrm{k}}^{*}(\boldsymbol{X}), L_{\mathrm{v}}^{*}(\boldsymbol{X}))), \quad (12)$$

where L_{q}^{*}, L_{k}^{*}, L_{v}^{*} and L_{out}^{*} are the pruned layers corresponding to the original layers L_{q}, L_{k}, L_{v} and L_{out}, respectively. The aforementioned pruning method is applied to these linear layers. It is important to note that, due to the unique nature of the attention mechanism, we select h parameters ($m = h$) in each iteration from the candidate set C, where h is the number of heads defined in the MHA.

2.2.3 Pruning of FFN module

The FFN module in the Transformer architecture can be represented as

$$\boldsymbol{Z} = \boldsymbol{Y} + L_2(L_1(\boldsymbol{Y})), \quad (13)$$

where $\boldsymbol{Z}$ is the output feature of the FFN module, and $\boldsymbol{Z} \in \mathbb{R}^{n\times d}$; L_1 and L_2 represent two output linear layers. The pruned representation of this module can be expressed as

$$\boldsymbol{Z} = \boldsymbol{Y} + L_2^{*}(L_1^{*}(\boldsymbol{Y})), \quad (14)$$

where L_1^{*} and L_2^{*} are the pruned layers corresponding to the original layers L_1 and L_2, respectively. The aforementioned pruning method is applied to these linear layers. Since these layers have a large number of neurons to be pruned, we select 32 parameters ($m = 32$) from the candidate set C in each iteration. The value of 32 is an empirical value, and further experimental explanations are provided.

2.3 Pruning algorithm

2.3.1 GP algorithm based on global optimization

Algorithm 2 illustrates the proposed GP algorithm based on global optimization. In this process, optimization is based on the global changes in the network. It applies the aforementioned greedy selection process to each layer designated for pruning. The input is the network to be pruned. For each layer in the network identified for pruning, the process outlined in Algorithm 1 is executed. The pruned layers obtained are then substituted for the original layers. Following this procedure, the pruned DETR-like network is obtained.

Algorithm 2: GP algorithm

Input: pretrained network D with L pruning layers.
Output: the pruned network D_{p}.
1: **for** $l = 1, 2, \cdots, L$ **do**
2: Apply Algorithm 1 to obtain the pruned layer L_l^{*}.
3: Replace the lth layer L_l of D with L_l^{*}.
4: **end for**

2.3.2 MGP algorithm based on global optimization

The algorithmic process of Algorithm 2 is straightforward and effective. However, further investigation reveals that in the Transformer architecture, the MHA module and the FFN module serve distinct functions. The attention mechanism in the MHA module primarily facilitates feature interaction, while the FFN module chiefly enhances the model's capacity for nonlinear transformations. Therefore, it is necessary to consider pruning the MHA module and the FFN module separately to prevent mutual interference. Table 1 shows the results of different pruning sequences for modules in each layer, where Params refers to the total amount of trainable parameters in the model and mAP refers to the mean average precision.

Table 1 Results of different pruning sequences for modules in each layer

Sequence	Params	mAP/%
ORIGIN	44×10^{6}	44.1
MHA-FFN	$\mathbf{34\times10^{6}}$	45.3
MHA-FFN-I	38×10^{6}	45.4
MHA-FFN-Alternate-I	37×10^{6}	**45.9**

As illustrated in Table 1, the following pruning sequences are evaluated.

1) ORIGIN: this sequence represents the original, unpruned network.

2) MHA-FFN: this sequence involves initially pruning the layers in the MHA module in their original order, followed by pruning the layers in the FFN module in the same order.

3) MHA-FFN-I: this sequence starts with pruning the layers in the MHA module in reverse order, then proceeds to prune the layers in the FFN module in reverse order.

4) MHA-FFN-Alternate-I: this sequence consists of pruning the layers in the MHA module in a reverse alternating order, followed by a similar reverse alternating order for pruning the layers in the FFN module.

Based on the observed compression effectiveness, the MHA-FFN sequence demonstrates the best results. Therefore, we select this sequence for designing the MGP algorithm. Algorithm 3 illustrates the algorithmic process, which involves initially pruning the layers within the MHA module, followed by pruning the layers within the FFN module. This sequence ultimately yields the pruned DETR-like network.

Algorithm 3: MGP algorithm

Input: pretrained network D with L pruning layers.
Output: the pruned network D_p.
1: **for** $l = 1, 2, \cdots, L$ **do**
2: Obtain the pruned layer $L_{\mathrm{M},l}^{*}$, if l belongs to MHA, apply Algorithm 1.

中国知网 https://www.cnki.net

(continued)

Algorithm 3: MGP algorithm

3: Replace the lth layer of the MHA in D with $L^*_{\mathrm{M},l}$.

4: **end for**

5: **for** $l = 1, 2, \cdots, L$ **do**

6: Obtain the pruned layer $L^*_{\mathrm{F},l}$, if l belongs to FFN, apply Algorithm 1.

7: Replace the lth layer of the FFN in D with $L^*_{\mathrm{F},l}$.

8: **end for**

3 Experimental section

3.1 Dataset and evaluation metrics

In this experiment, we utilize the COCO 2017 dataset, which is one of the most widely used datasets in the field of object detection. It is composed of images with 80 categories. These images are divided into train2017 with 118 000 images, val2017 with 5 000 images, and test2017 with 41 000 images. In experiments, it is a common convention to train models on the train2017 and evaluate on the val2017. We report the results on the COCO 2017 validation dataset under different intersection over union (IoU) thresholds and object scales.

Additionally, regarding the metrics for measuring model complexity, we use the two most common indicators: the Params and the number of floating point operations (FLOPs).

3.2 Implementation details

We choose the DN-DAB-DETR[5] as the baseline DETR network for pruning and select the ResNet-50 model as the backbone network. For hyperparameters, we use a 6-layer Transformer encoder and a 6-layer Transformer decoder, both with a hidden dimension of 256. For the learning rate (LR) scheduler, we use an initial LR of 1×10^{-4}, and reduce the LR at the 40th epoch by multiplying it by 0.1. We use the AdamW optimizer with a weight decay of 1×10^{-4}. The batch size is eight. All experiments are conducted on four Nvidia RTX4090 GPUs.

We add eight parameters (head number) to $\boldsymbol{V}_{\mathrm{cs}}$ for each MHA module and 32 parameters to $\boldsymbol{V}_{\mathrm{cs}}$ for each FFN module. For GP algorithm, the accuracy target is initialized as the accuracy of the original model, and after each layer is pruned, it decreases by 0.5%. The pruning process for a layer is considered complete when the target accuracy of 98.0% is reached. For MGP algorithm, the accuracy target for each layer is set to 98.0% of the accuracy of the previous layer.

4 Results and Discussion

4.1 State-of-the-art comparison

As shown in Table 2, we compare the experimental results of MGP algorithm with DETR-R50[1], DETR-DC5-R50[1] and some of the most recent DETR-like methods, including Deform-DETR-R50[2], Co-DETR-R50[3], DAB-DETR-R50[4], DN-DAB-DETR-R50[5], DN-Deform-DETR-R50[5] and Pruning DETR[12].

Table 2 Results for our method and other detection models under the same setting

Method	Epochs	Params	GFLOPs	mAP/%	mAP_{50}/%	mAP_{75}/%	$\mathrm{mAP}_{\mathrm{S}}$/%	$\mathrm{mAP}_{\mathrm{M}}$/%	$\mathrm{mAP}_{\mathrm{L}}$/%
DETR-R50[1]	500	41×10^6	86	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5-R50[1]	500	41×10^6	187	43.3	63.1	45.9	22.5	47.3	61.1
Deform-DETR-R50[2]	50	40×10^6	173	43.8	62.6	47.7	26.4	47.1	58.0
Co-DETR-R50[3]	50	44×10^6	90	40.9	61.8	43.3	20.8	44.6	59.2
DAB-DETR-R50[4]	50	44×10^6	94	42.2	63.1	44.7	21.5	45.7	60.3
DN-DAB-DETR-R50[5]	50	44×10^6	94	44.1	64.4	46.7	22.9	48.0	63.4
DN-Deform-DETR-R50[5]	50	48×10^6	195	43.4	61.9	47.2	24.8	46.8	59.4
Pruning DETR[12]	50	$\mathbf{26\times10^6}$	126	42.7	62.8	45.0	20.5	45.6	60.9
MGP (ours)	50	34×10^6	**81**	**45.3**	65.5	48.2	25.3	49.2	64.2

Notes: mAP is at IoU thresholds ranging from 0.50 to 0.95 in increments of 0.05 (primary challenge metric); mAP_{50} refers to mAP at IoU threshold of 0.50; mAP_{75} refers to mAP at IoU threshold of 0.75; $\mathrm{mAP}_{\mathrm{S}}$ refers to mAP for small objects; $\mathrm{mAP}_{\mathrm{M}}$ refers to mAP for medium objects; $\mathrm{mAP}_{\mathrm{L}}$ refers to mAP for large objects.

Compared to other methods, our method reduces the Params to 34×10^6 and GFLOPs (1 GFLOPs = 10^9 FLOPs) to 81, while improving mAP to 45.3%. As more than half of the parameters in DN-DETR are located in the backbone network and our method focuses solely on the Transformer part, the pruning results may not appear as significant in the overall context. In Section 4.2, we present more substantial experimental comparisons.

4.2 GP vs. MGP

To better illustrate the differences between our two proposed algorithms, we conduct comparative experiments. As shown in Table 3, compared to the baseline network DN-DAB-DETR-R50[5], the GP algorithm reduces the Params from 44×10^6 to 36×10^6 and GFLOPs from 94 to 83 while increasing mAP from 44.1% to 45.4%. With the MGP algorithm, Params

中国知网 https://www.cnki.net

is reduced from 44×10^6 to 34×10^6, GFLOPs is reduced from 94 to 81, and mAP is increased from 44.1% to 45.3%. The results indicate that, while maintaining nearly identical accuracy, the MGP algorithm achieves a higher parameter compression ratio compared to the GP algorithm.

Table 3 Comparison results of two methods under the same setting

Method	Epochs	Params	GFLOPs	mAP/%	mAP_{50}/%	mAP_{75}/%	mAP_S/%	mAP_M/%	mAP_L/%
DN-DAB-DETR-R50[5]	500	44×10^6	94	44.1	64.4	46.7	22.9	48.0	63.4
GP (ours)	500	36×10^6	83	**45.4**	65.6	48.7	24.8	49.3	64.0
MGP (ours)	50	**34×10^6**	**81**	45.3	65.5	48.2	25.3	49.2	64.2

4.3 Comparison of pruning Transformer components

The DN-DETR network primarily comprises two key components: the backbone network for feature extraction and the Transformer architecture for feature modeling. Our pruning algorithm is specifically designed for the Transformer architecture. Table 4 provides a more detailed view of the pruning effects of our two algorithms on the overall Transformer architecture, as well as its encoder and decoder components separately. In the overall Transformer architecture, the GP algorithm reduces Params by 36% and GFLOPs by 31%, while the MGP algorithm reduces Params by 49% and GFLOPs by 44%. These results indicate that a considerable number of parameters in the Transformer architecture of the large DN-DETR network are indeed not contributing effectively. Furthermore, the pruning results in the Transformer encoder and decoder components show that a higher proportion of parameters are eliminated in the encoder component. This is because the encoder is more focused on feature modeling, while the decoder is more concerned with feature interaction.

Table 4 Comparison results of two algorithms on DN-DETR's Transformer, encoder and decoder

Architecture	Params	GFLOPs
Transformer[5]	18.5×10^6	11.3
Transformer (GP)	11.9×10^6 (↓36%)	7.8 (↓31%)
Transformer (MGP)	**9.5×10^6** (↓**49**%)	**6.3** (↓**44**%)
Transformer encoder[5]	6.8×10^6	6.4
Transformer encoder (GP)	4.3×10^6 (↓37%)	4.2 (↓34%)
Transformer encoder (MGP)	3.1×10^6 (↓54%)	3.2 (↓50%)
Transformer decoder[5]	11.7×10^6	4.9
Transformer decoder (GP)	7.6×10^6 (↓35%)	3.6 (↓27%)
Transformer decoder (MGP)	6.4×10^6 (↓45%)	3.2 (↓35%)

4.4 Ablation study for *m*

In the previously mentioned Eqs. (12) and (14), we used different parameter settings for m, meaning we retained a varying number of parameters each time during the greedy process for layers within the MHA and FFN modules. In Table 5, we demonstrate the impact of different parameter settings for m on the pruning algorithm, with all experiments following the algorithm of MGP.

Table 5 Comparison results of different parameter settings for m based on advanced experiments

Row	M_m	F_m	Params	GFLOPs	mAP
1	4	16	37×10^6	83	**45.5**
2	8	32	**34×10^6**	**81**	45.3
3	16	64	36×10^6	82	45.4

Notes: M_m refers to the configuration of the MHA module; F_m refers to the configuration of the FFN module.

It can be seen from Table 5 that selecting different parameter settings for m has varying impacts on the outcomes. It's crucial to choose an appropriate value for m to achieve optimal results. The selection of m must strike a balance to ensure effective pruning while maintaining or enhancing the network's performance.

4.5 Pruning ratios of each layer

To delve deeper into the pruning of different layers in DN-DETR, we present an analysis of the pruning ratios in the Transformer encoder and decoder of DN-DETR, based on the experiments conducted using the MGP algorithm. Figure 4 illustrates the pruning ratios of each layer in the Transformer encoder of DN-DETR. Figure 4(a) shows the pruning ratios for each layer in the MHA module, where each original layer has 256 neurons. Figure 4(b) shows the pruning ratios for each layer in the FFN module, where each original layer has 2 048 neurons. The results indicate that the earlier layers in the encoder contain more redundant parameters.

中国知网 https://www.cnki.net

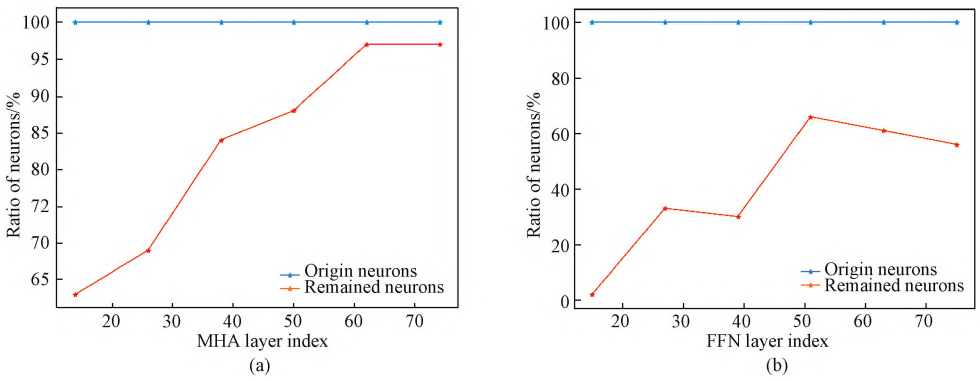

Fig. 4 Pruning ratios of each layer in DN-DETR's Transformer encoder: (a) MHA; (b) FFN

As shown in Fig. 5, the pruning ratios of each layer in the Transformer decoder of the DN-DETR are displayed. Figure 5(a) shows the pruning ratios for each layer in the MHA module, where each original layer has 256 neurons. Figure 5(b) shows the pruning ratios for each layer in the FFN module, where each original layer has 2 048 neurons. The irregular results indicate that each layer in the decoder serves a distinct function.

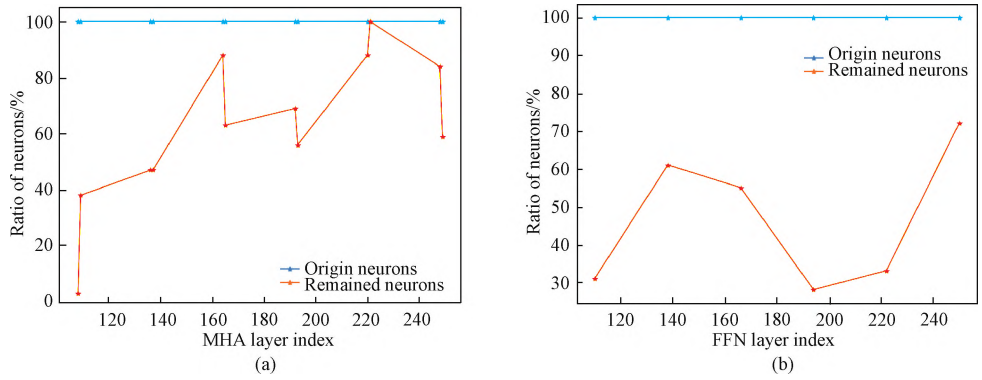

Fig. 5 Pruning ratios of each layer in DN-DETR's Transformer decoder: (a) MHA; (b) FFN

5 Conclusions

This research diverges from the majority of recent efforts that aimed at improving the DETR network through the addition of auxiliary modules. Instead, it innovatively enhances the performance of the DETR network through pruning. We propose a GP algorithm based on global optimization that iteratively searches and retains useful parameters in the DETR-like network. Further, recognizing the distinct functions of the MHA and FFN modules in the Transformer architecture, we propose an MGP algorithm, separately pruning these two modules. Experimental results prove that the GP and the MGP algorithms achieve superior performance. The proposed method is versatile, applicable to most DETR networks, and paves the way for further in-depth research in the future.

References

[1] CARION N, MASSA F, SYNNAEVE G, et al. End-to-end object detection with transformers [C]//European Conference on Computer Vision. Cham: Springer, 2020: 213-229.

[2] ZHU X Z, SU W J, LU L W, et al. Deformable DETR: deformable transformers for end-to-end object detection[EB/OL]. (2021-03-18)[2024-01-21]. https://arxiv.org/abs/2010.04159.

[3] MENG D P, CHEN X K, FAN Z J, et al. Conditional DETR for fast training convergence [C]//2021 IEEE/CVF International Conference on Computer Vision (ICCV). New York: IEEE, 2021: 3631-3640.

[4] LIU S L, LI F, ZHANG H, et al. DAB-DETR: dynamic anchor boxes are better queries for DETR[EB/OL]. (2022-05-30)[2023-12-21]. https://arxiv.org/abs/2201.12329v1.

[5] LI F, ZHANG H, LIU S L, et al. DN-DETR: accelerate DETR training by introducing query denoising [C]//2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Los Alamitos: IEEE, 2022: 13609-13617.

[6] YE M, WU L M, LIU Q. Greedy optimization provably wins the lottery: logarithmic number of winning tickets is enough [J]. *Advances in Neural Information Processing Systems*, 2020, 33: 16409-16420.

中国知网 https://www.cnki.net

[7] ROH B, SHIN J, SHIN W, et al. Sparse DETR: efficient end-to-end object detection with learnable sparsity [EB/OL]. (2022-05-04) [2023-12-21]. https://arxiv.org/abs/2111.14330v1.
[8] HAN S, POOL J, TRAN J, et al. Learning both weights and connections for efficient neural network[C]. 29th Annual Conference on Neural Information Processing Systems (NIPS). La Jolla: NIPS, 2015, 28.
[9] LIU Z, LI J G, SHEN Z Q, et al. Learning efficient convolutional networks through network slimming [C]//2017 IEEE International Conference on Computer Vision (ICCV). New York: IEEE, 2017: 2755-2763.
[10] DOSOVITSKIY A, BEYER L, KOLESNIKOV A, et al. An image is worth 16×16 words: transformers for image recognition at scale[C]// 35th Annual Conference on Neural Information Processing Systems (NIPS). La Jolla: NIPS, 2021.
[11] ZHU M J, TANG Y H, HAN K. Vision transformer pruning [EB/OL]. (2021-08-14) [2023-12-21]. https://arxiv.org/abs/2104.08500.
[12] SUN H Y, ZHANG S L, TIAN X, et al. Pruning DETR: efficient end-to-end object detection with sparse structured pruning [J]. *Signal, Image and Video Processing*, 2024, 18(1): 129-135.

基于全局优化的 DETR 架构网络贪心剪枝算法

黄秋波*, 徐敬赛, 张亚魁, 王 梅, 陈德华
东华大学 计算机科学与技术学院, 上海 201620

摘 要: 目标检测 Transformer (detection Transformer, DETR) 成功地定义了 Transformer 架构在目标检测领域的范式, 其端对端检测的流程和集合预测的思想成为近些年最热门的网络架构之一。对 DETR 的改进工作层出不穷。然而, DETR 及其变种需较多的内存资源和计算消耗, 这些网络巨大的参数量对于模型的部署十分不利。为解决该问题提出一种贪心剪枝算法, 运用于一个变种 DN-DETR 上, 可剔除 DN-DETR 中 Transformer 架构的冗余参数。考虑到 Transformer 架构中多头注意力 (multi-head attention, MHA) 模块和前馈网络 (feed-forward network, FFN) 模块的不同作用, 进一步提出一种分模块贪心剪枝算法, 将两个模块分开考虑, 应用各自最优的策略与参数。在 COCO 2017 数据集上验证了所提方法的有效性。通过分模块贪心剪枝算法得到的模型, 对比 DN-DETR 的 Transformer 架构, 参数量减少了 49%, 浮点运算次数减少了 44%, 同时模型的平均精度由 44.1%提高到 45.3%。

关键词: 模型剪枝; DETR; Transformer 架构; 目标检测

中国知网 https://www.cnki.net