PRUNING PARAMETERIZATION WITH BI-LEVEL OPTIMIZATION FOR EFFICIENT SEMANTIC SEGMENTATION ON THE EDGE

Anonymous authors
Paper under double-blind review

ABSTRACT

With the ever-increasing popularity of edge devices, it is necessary to implement real-time segmentation on the edge for autonomous driving and many other applications. Vision Transformers (ViTs) have shown considerably stronger results for many vision tasks. However, ViTs with the full-attention mechanism usually consume a large number of computational resources, leading to difficulties for real-time inference on edge devices. In this paper, we aim to derive ViTs with fewer computations and fast inference speed to facilitate the dense prediction of semantic segmentation on edge devices. To achieve this, we propose a pruning parameterization method to formulate the pruning problem of semantic segmentation. Then we adopt a bi-level optimization method to solve this problem with the help of implicit gradients. Our experimental results demonstrate that we can achieve 38.9 mIoU on ADE20K val with a speed of 56.5 FPS on Samsung S21, which is the highest mIoU under the same computation constraint with real-time inference.

1 INTRODUCTION

Inspired by the extraordinary performance of Deep Neural Networks (DNNs), DNNs have been applied to various tasks. In this work, we focus on semantic segmentation, which aims to assign a class label to each pixel of an image to perform a dense prediction. It plays an important role in many real-world applications, such as autonomous driving. However, as a dense prediction task, segmentation models usually have complicated multi-scale feature fusion structures with large feature sizes, leading to tremendous memory and computation overhead with slow inference speed.

To reduce the memory and computation cost, certain lightweight CNN architectures (Li et al., 2020b; Yu et al., 2021; Fan et al., 2021) are designed for efficient segmentation. Besides CNNs, inspired by the recent superior performance of vision transformers (ViTs) (Dosovitskiy et al., 2021), some works (Zheng et al., 2021; Cheng et al., 2021b;a) adopt ViTs in segmentation tasks to explore self-attention mechanism with the global receptive field. However, it is still difficult for ViTs to reduce the computation cost of the dense prediction for segmentation with large feature sizes.

With the wide spread of edge devices such as mobile phones, it is essential to perform real-time inference of segmentation on edge devices in practical applications. To facilitate mobile segmentation, the state-of-the-art work TopFormer Zhang et al. (2022) adopts a token pyramid transformer to produce scale-aware semantic features with tokens from various scales. It significantly outperforms CNN- and ViT-based networks across different semantic segmentation datasets and achieves a good trade-off between accuracy and latency. However, it only partially optimizes the token pyramid module, which costs most of the computations and latency.

In this work, we propose a pruning parameterization method with bi-level optimization to further enhance the performance of TopFormer. Our objective is to search for a suitable layer width for each layer in the token pyramid module, which is the main cost of computations and latency (over 60%). To achieve this, we first formulate the problem with pruning parameterization to build a pruning framework with a soft mask as a representation of the pruning policy. With this soft mask, we further adopt thresholding to convert the soft mask into a binary mask so that the model is trained with actual pruned weights to obtain pruning results directly. This is significantly different from other methods (Liu et al., 2017; Guan et al., 2022) to train with unpruned small non-zero weights and use fine-tuning

to mitigate the performance degradation after applying pruning. Besides, to update the soft mask as long as the pruning policy, we adopt to straight though estimator (STE) method to make the soft mask differentiable. Thus, we can build the pruning parameterization framework with minimal overhead.

Based on this framework, we need to search the best-suited layer width for each layer in the token pyramid module. It is non-trivial to perform the search. As the token pyramid module needs to extract multi-scale information from multiple spatial resolutions, the large hierarchical search space leads to difficulties of convergence. To resolve this problem, we adopt a bi-level optimization method. In the outer optimization, we try to obtain the pruning policy based on the pruning parameters (the soft mask). In the inner optimization, the optimized model weights with the best segmentation performance under this soft mask can be obtained. Compared with a typical pruning method, our work incorporates the implicit gradients with second-order derivatives to further guide the update of the soft mask and achieve better performance. Our experimental results demonstrate that we can achieve 38.9 mIoU (mean class-wise intersection-over-union) on the ADE20K dataset with a speed of 56.5 FPS on Samsung S21, which is the highest mIoU under the same computation constraint with real-time inference speed. We summarize our contributions below,

- We propose a pruning parameterization method to build a pruning framework with a soft mask. We further use a threshold-based method to convert the soft mask into the binary mask to perform actual pruning during model training and inference. Besides, STE is adopted to update the soft mask efficiently through gradient descent optimizers.
- To solve the pruning problem formulated with the framework of pruning parameterization, we propose a bi-level optimization method to utilize implicit gradients for better results. We show that the second-order derivatives in the implicit gradients can be efficiently obtained through first-order derivatives, saving computations and memory.
- Our experimental results demonstrate that we can achieve the highest mIoU under the same computation constraint on various datasets. Specifically, we can achieve 38.9 mIoU on the ADE20K dataset with a real-time inference speed of 56.5 FPS on the Samsung S21.

2 Related Work

Real-Time Semantic Segmentation. Though semantic segmentation based on CNNs (Zhao et al., 2017; Chen et al., 2018; Fu et al., 2019; Huang et al., 2019) can achieve great performance, it typically costs large amounts of computations with slow inference speed. Furthermore, with the ever-increasing popularity of edge devices such as mobile phones, it is necessary to achieve fast inference speed for semantic segmentation on edge devices. Besides human designed lightweight models (Yu et al., 2018; Li et al., 2020b; Fan et al., 2021), neural architecture search (NAS) methods (Liu et al., 2019; Li et al., 2019c; Chen et al., 2019; Zhang et al., 2021) are also adopted to search lightweight models.

As a pioneer in handcrafted real-time segmentation, ENet (Paszke et al., 2016) designs a lightweight model for fast inference. DeepLabV3+ (Chen et al., 2018) adopts atrous separable convolution to reduce computation counts and uses the lightweight MobileNetV2 (Sandler et al., 2018) as the backbone. BiSeNet (Yu et al., 2018; 2021) and STDC (Fan et al., 2021) utilizes a two-branch architecture, where the deep branch extracts spatial information, and the shallow branch learns details. SFNet (Li et al., 2020b) uses flow alignment module to fuse context and spatial information.

Inspired by the recent success of NAS, some works automatically search lightweight segmentation models. Auto-DeepLab (Liu et al., 2019) searches a model with extremely high segmentation performance regardless of the computation budgets or overhead. To reduce the computation cost, FasterSeg (Chen et al., 2019) incorporates latency regularization during search. DCNAS (Zhang et al., 2021) uses a densely connected search space and employs a gradient-based direct search method. NASViT (Gong et al., 2021) proposes a gradient projection algorithm to deal with the gradient conflict issues and improve the convergence performance. HR-NAS (Ding et al., 2021a) keeps high-resolution representations in its encoder and the search process to maintain high accuracy in dense prediction.

Although many lightweight segmentation models are developed, it is still hard for them to run real-time inference on resource- and power-limited GPUs of edge devices.

Vision Transformers. By utilizing the self-attention mechanism, ViTs (Dosovitskiy et al., 2021) can achieve competitive results against CNN models in vision tasks. Inspired by the great success

of ViTs, many research efforts are devoted to dense prediction tasks with ViTs on complex datasets. SETR (Zheng et al., 2021) utilizes a ViT-based encoder to extract high-level semantic information. Instead of per-pixel prediction, MaskFormer (Cheng et al., 2021b) performs mask-based prediction with a customize backbone and a transformer-based decoder. With a transformer decoder to explore masked attention, mask2Former (Cheng et al., 2021a) proposes a universal architecture and achieves SOTA performance for various segmentation tasks. MobileViT (Mehta & Rastegari, 2021) and MobileFormer (Chen et al., 2021) mix CNN and ViT in their architectures, but they are not fast enough to achieve real-time inference on edge devices. TopFormer (Zhang et al., 2022) utilizes the CNN-based token pyramid module to extract multi-level tokens and lightweight ViT blocks to extract semantic information. It can achieve low latency and high accuracy on complex datasets.

Neural Architecture Search and Pruning. NAS and pruning are commonly used to find lightweight models and reduce the computation overhead. Given a search space, NAS tries to identify a superior model automatically. Reinforcement learning (RL) based NAS (Zoph & Le, 2017; Pham et al., 2018) and evolution-based NAS (Elsken et al., 2018; Real et al., 2019) usually need to train and evaluate each candidate model, leading to tremendous searching cost. To mitigate this, gradient-based NAS (Liu et al., 2018; Cai et al., 2018; Chu et al., 2019; Guo et al., 2020b) is proposed to formulate a supernet with all candidate architectures and search the outstanding architecture through gradient descent methods. But it costs huge memory to incorporate all candidate architectures.

Network pruning is a compression technique to effectively reduce the DNN storage and computation cost. In this work, we focus on structured pruning (Wen et al., 2016; Li et al., 2019a) to remove entire filters or channels of CONV layers, which can be accelerated effectively for fast inference.

3 Problem Formulation with Pruning Parameterization

In our method, we first formulate the pruning problem with pruning parameterization. Previous pruning methods usually depend on the magnitudes of model weights and adopt the in-differentiable sorting operations, leading to inconsistent performance after applying pruning with sorting and additional overhead with fine-tuning. To mitigate this, we propose a pruning parameterization method, which uses a soft mask (rather than magnitudes of weights) to indicate whether to prune and get rid of sorting operations. With the STE method (Bengio et al., 2013), we are able to represent the pruning with parameters and directly train the pruning like a typical model training. Then we formulate our problem with pruning parameterization and introduce our solution.

3.1 Soft Mask Construction

We first introduce how to construct the soft mask. To accelerate the inference, we adopt channel pruning to search for a suitable width for each convolution (CONV) layer. Specifically, we insert a depth-wise CONV layer following each CONV layer that is supposed to be pruned as below,

$$\boldsymbol{a}_l = \boldsymbol{s}_l \odot (\boldsymbol{w}_l \odot \boldsymbol{a}_{l-1}), \tag{1}$$

where $\odot$ denotes the convolution operation. $\boldsymbol{w}_l \in R^{o\times i\times k\times k}$ is the weight parameters in l-th CONV layer, with o output channels, i input channels, and kernels of size $k \times k$. $\boldsymbol{a}_l \in R^{n\times o\times t\times t'}$ represents the output features of the l-th layer, with o channels and $t \times t'$ feature size. n denotes batch size. $\boldsymbol{s}_l \in R^{o\times 1\times 1\times 1}$ is the weights of the depth-wise CONV layer. Each output channel of $\boldsymbol{w}_l \odot \boldsymbol{a}_{l-1}$ corresponds to one single element of $\boldsymbol{s}_l$. Thus $\boldsymbol{s}_l$ can serve as a soft mask or pruning indicator for the l-th CONV layer to indicate whether to prune the corresponding output channels.

3.2 Forward and Backward Propagation

Since $\boldsymbol{s}_l$ is only a soft mask with continuous values, it can not represent the binary operation of pruning. To solve this, we adopt a threshold, and the forward pass with the mask is represented as

$$\boldsymbol{b}_l = \begin{cases} 1, \boldsymbol{s}_l > \tau \\ 0, \boldsymbol{s}_l \leq \tau \end{cases} \quad \text{(element-wise)}, \qquad \boldsymbol{a}_l = \boldsymbol{b}_l \odot (\boldsymbol{w}_l \odot \boldsymbol{a}_{l-1}), \tag{2}$$

where $\boldsymbol{b}_l \in \{0, 1\}^{o\times 1\times 1\times 1}$ is the binarized $\boldsymbol{s}_l$, and τ is a threshold which is simply set to 0.5 in our case. Each element of $\boldsymbol{b}_l$ corresponds to one output channel of $\boldsymbol{w}_l \odot \boldsymbol{a}_{l-1}$. In the forward pass, we

first convert the soft mask into a binary mask through a threshold and then perform the depth-wise CONV to perform actual pruning. Thus the output channels corresponding to the zero elements in $\boldsymbol{b}_l$ are pruned, and the rest channels are preserved. We show the proof in Appendix A.

The binary operation in Equation (2) is non-differential, leading to difficulties for back-propagation. To solve this, we propose to incorporate STE (Bengio et al., 2013) for back-propagation as below,

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{s}_l} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_l}, \tag{3}$$

where we directly pass the gradients of $\boldsymbol{b}_l$ to $\boldsymbol{s}_l$ so that we can update the soft mask.

With the binarization and STE method, the pruning process can be represented with the soft mask $\boldsymbol{s} = \{\boldsymbol{s}_l\}$. We can update the soft mask and its corresponding binary mask to update the pruning policy based on its gradients. Thus $\boldsymbol{s}$ is the pruning parameters to denote and control pruning.

Difference with other pruning works. Based on pruning parameterization, we decouple the pruning policy from model parameter magnitudes so that pruning does not further depend on the weight magnitudes. Unlike previous works on pruning to force pruned weights to be or as close as to zeros (Liu et al., 2017; He et al., 2017; Guan et al., 2022), our method does not have such a constraint that pruned weights should be zero. Instead, once the corresponding binary mask is turned from 1 to 0, the information in pruned channels is preserved rather than zeroed out since zeros in $\boldsymbol{b}_l$ can block gradient flow to the corresponding weights. As a result, pruned channels are free to recover and contribute to accuracy if their corresponding elements in $\boldsymbol{b}_l$ switch from 0 to 1.

Difference with other mask methods. Although some other works also adopt indicator/mask-based pruning such as (Guan et al., 2022; Kim et al., 2020; Guo et al., 2020a; Kang & Han, 2020), our method is more straightforward and effective. For example, unlike our method to train the soft mask with STE directly, DAIS (Guan et al., 2022) relaxes the binarized channel indicators to be continuous. To bridge the non-negligible discrepancy between the continuous model and the target binarized model, it further uses an annealing-based procedure to steer the indicator convergence toward binarized states. Some works (Kang & Han, 2020; You et al., 2019) adopts batchnorm (BN) layers and uses the cumulative density function (CDF) of a Gaussian distribution as the mask variable, with a CDF-based loss function and the Gumbel-Softmax trick to update the mask at the cost of additional random sampling and complex gradient revision. Besides, the works (Gao et al., 2020; You et al., 2019) create a mask to multiple the channels weights. This is different from our design with the depth-wise CONV operation for easy mask creation and direct mask training.

3.3 Training Loss with Pruning Parameterization

Based on the soft mask, we can train and prune the model with the following loss function,

$$\mathcal{L}_m(\boldsymbol{w}, \boldsymbol{s}) = \mathcal{L}(\boldsymbol{w}, \boldsymbol{s}) + \beta \cdot \mathcal{L}_{reg}(\boldsymbol{s}), \tag{4}$$

where $\mathcal{L}(\boldsymbol{w}, \boldsymbol{s})$ is the cross-entropy loss, and $\mathcal{L}_{reg}$ is the regularization term related to the sparsity or pruning ratio. For simplicity, we take Multiply-Accumulate operations (MACs) as the regularization rather than parameter number to estimate the on-device execution cost more precisely. β can weight the loss and stabilize training. $\mathcal{L}_{\text{reg}}$ can be defined as the squared ℓ_2 norm of the difference between current MACs and target MACs $\mathcal{C}$,

$$\mathcal{L}_{\text{reg}} = \left| \sum_l o'_l \times i_l \times t_l \times t'_l \times k^2 - \mathcal{C} \right|^2, \tag{5}$$

where o'_l is the number of remaining channels after pruning, and $t_l \times t'_l$ is the output feature size.

3.4 Problem Formulation

With pruning parameterization, the per-layer width search problem can be formulated as follows

$$\min_{\boldsymbol{s}} \quad \mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s}), \tag{6}$$

$$\text{s.t.} \quad \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \quad \mathcal{L}(\boldsymbol{w}, \boldsymbol{s}) + \frac{1}{2\lambda} \|\boldsymbol{w}\|_2^2. \tag{7}$$

It is a bi-level optimization problem (Gould et al., 2016). In the inner optimization of Equation (7), we optimize the model parameters under a given soft mask with a commonly used squared ℓ_2 norm as a regularization to mitigate the overfitting problem. In the outer optimization of Equation (6), we optimize the soft mask to minimize the loss. Each time before updating $\boldsymbol{s}$, we first need to obtain $\boldsymbol{w}^*$.

4 PROPOSED METHOD WITH BI-LEVEL OPTIMIZATION

The objective is to find the soft mask (search a suitable width) for each layer to minimize the loss with optimized model weights. We adopt a bi-level pruning method to solve this problem. Compared with a typical gradient descent method to update the parameters with first-order derivatives, the bi-level optimization method incorporates implicit gradients with second-order derivatives to adjust the first-order term, leading to higher training efficiency with better convergence results. For easy of expression, since each channel has a corresponding mask, in this section, we broadcast the masks $\boldsymbol{s}$ and $\boldsymbol{b}$ so that the masks have the same dimension as the model weights $\boldsymbol{w}$.

4.1 BI-LEVEL OPTIMIZATION METHOD WITH IMPLICIT GRADIENTS

From the inner optimization, $\boldsymbol{w}^*$ is a function of $\boldsymbol{s}$ and different $\boldsymbol{s}$ can lead to different $\boldsymbol{w}^*$. Thus, to minimize $\mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s})$ in Problem (6), we need to compute the gradients with reference to $\boldsymbol{s}$ as below,

$$\frac{d\mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s})}{d\boldsymbol{s}} = \frac{d\boldsymbol{w}^*}{d\boldsymbol{s}} \nabla_{\boldsymbol{w}} \mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s}) + \nabla_{\boldsymbol{s}} \mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s}), \tag{8}$$

where $\nabla_{\boldsymbol{w}}$ and $\nabla_{\boldsymbol{s}}$ denote the partial derivatives of the loss function with reference to $\boldsymbol{w}$ and $\boldsymbol{s}$, respectively. $\frac{d\boldsymbol{w}^*}{d\boldsymbol{s}}$ represents the vector-wise full derivative, and we omit the transpose expression. Since $\boldsymbol{w}^*$ is implicitly defined as an optimization problem in Equation (7), $\frac{d\boldsymbol{w}^*}{d\boldsymbol{s}}$ is also known as implicit gradients (Rudin et al., 1976; Samuel & Tappen, 2009; Rajeswaran et al., 2019). With $g(\boldsymbol{w}, \boldsymbol{s}) = \mathcal{L}(\boldsymbol{w}, \boldsymbol{s}) + \frac{1}{2\lambda}\boldsymbol{w}^T\boldsymbol{w}$, $\frac{d\boldsymbol{w}^*}{d\boldsymbol{s}}$ can be obtained through the following,

$$\frac{d\boldsymbol{w}^*}{d\boldsymbol{s}} = -\nabla^2_{\boldsymbol{sw}} g(\boldsymbol{w}^*, \boldsymbol{s}) \nabla^2_{\boldsymbol{w}} g(\boldsymbol{w}^*, \boldsymbol{s})^{-1}, \tag{9}$$

where $\nabla^2_{\boldsymbol{sw}}$ and $\nabla^2_{\boldsymbol{w}}$ are the second-order partial derivatives. We show the proof in Appendix B.

The Hessian matrix $\nabla^2_{\boldsymbol{w}} g(\boldsymbol{w}^*, \boldsymbol{s})$ can be given by

$$\nabla^2_{\boldsymbol{w}} g(\boldsymbol{w}^*, \boldsymbol{s}) = \nabla^2_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{s}) + \frac{1}{\lambda}\mathbb{I} = \frac{1}{\lambda}\mathbb{I}, \tag{10}$$

where we adopt a Hessian-free approximation that $\nabla^2_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{s}) = 0$ as DNNs usually have piece-wise linear decision boundary with ReLU functions. Thus, Equation (8) can be transformed to

$$\frac{d\mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s})}{d\boldsymbol{s}} = \nabla_{\boldsymbol{s}} \mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s}) - \lambda \nabla^2_{\boldsymbol{sw}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{s}) \nabla_{\boldsymbol{w}} \mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s}). \tag{11}$$

Compared with a typical gradient descent method, the bi-level optimization incorporates the second-order derivatives $\nabla^2_{\boldsymbol{sw}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{s}) \nabla_{\boldsymbol{w}} \mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s})$ to adjust the first-order term $\nabla_{\boldsymbol{s}} \mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s})$.

It is difficult to obtain the second-order derivatives $\nabla^2_{\boldsymbol{sw}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{s})$. Usually certain approximation methods such as finite difference (Chen et al., 2020; Fallah et al., 2020) may be adopted to save computation cost. But here we show that in this specific problem, we can directly obtain the analytical solution with first-order derivatives, which greatly saves computation efforts without approximation. Note that each channel has its corresponding mask, and we denote the masked channels as $\boldsymbol{w}_b = \boldsymbol{w}^* \cdot \boldsymbol{b}$ where $\cdot$ means the element-wise multiplication and $\boldsymbol{b}$ is defined in Equation (2). We can obtain that

$$\nabla^2_{\boldsymbol{sw}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{s}) = \text{diag}(\nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b)) \tag{12}$$

where $\text{diag}(\cdot)$ represents formulating a diagonal matrix with the diagonal vector. We show the proof in Appendix C. Then we can transform Equation (11) into the following,

$$\frac{d\mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s})}{d\boldsymbol{s}} = \nabla_{\boldsymbol{s}} \mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s}) - \lambda \nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b) \cdot \nabla_{\boldsymbol{w}} \mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s}). \tag{13}$$

Thus although the implicit gradients incorporate second-order derivatives, it can be analytically expressed with first-order derivatives, greatly saving computation cost without any approximations.

In Equation (13), we need to create two copies $\boldsymbol{w}_b$ and $\boldsymbol{w}$ to obtain their derivatives, which are still not memory efficient. To further reduce the complexity, we can obtain $\nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b)$ in the following,

$$\nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b) = \begin{cases} \nabla_{\boldsymbol{w}} \mathcal{L}_m(\boldsymbol{w}), & \boldsymbol{b} = 1 \\ 0, & \boldsymbol{b} = 0 \end{cases} \quad \text{(element-wise)} \tag{14}$$

The pruned weights ($\boldsymbol{b} = 0$) do not contribute to the loss, so their gradients are zero. The difference between $\mathcal{L}(\cdot)$ and $\mathcal{L}_m(\cdot)$ is just the regularization term $\mathcal{L}_{\text{reg}}(\cdot)$, which only relates to the sparsity and does not care the weight values. So $\nabla_{\boldsymbol{w}_b}\mathcal{L}(\boldsymbol{w}_b)$ and $\nabla_{\boldsymbol{w}}\mathcal{L}_m(\boldsymbol{w})$ are equal if their weights are not pruned ($\boldsymbol{b} = 1$). Combining Equation (13) and (14), we can obtain the following,

$$\frac{d\mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s})}{d\boldsymbol{s}} = \nabla_{\boldsymbol{s}}\mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s}) - \lambda \boldsymbol{b} \cdot \left[\nabla_{\boldsymbol{w}}\mathcal{L}_m(\boldsymbol{w}^*, \boldsymbol{s})\right]^2 . \tag{15}$$

We can see that there is no need to keep a copy and compute the gradients of $\boldsymbol{w}_b$, thus saving memory. During practical implementation, since $\boldsymbol{s}$ and $\boldsymbol{b}$ are the channel-wise masks, we accumulate the gradients of all weights in each channel to update the channel mask following Equation (15).

4.2 Bi-level Optimization Framework

In each iteration, our framework has two steps, including the model weights training step and the mask updating step. In the first step, we update the weights $\boldsymbol{w}$ with a few training steps for a fixed mask $\boldsymbol{s}$. Next in the second step, we update $\boldsymbol{s}$ with implicit gradients following Equation (15). Then we move on to the next iteration. In each step, we only update $\boldsymbol{w}$ or $\boldsymbol{s}$ without changing the other parameter. We discuss the advantages of the proposed method in the following.

No Need for Finetuning. After training with the proposed method, we can obtain the final layer-wise pruning policy following $\boldsymbol{s}$ and the sparse model. Since the model is already trained with the binary masks, it can achieve good segmentation performance without further fine-tuning.

First-Order Optimization. During the computation, we only adopt first-order optimization. Though we incorporate second-order derivatives in implicit gradients, we show that it can be analytically expressed with first-order derivatives in Equation (12), which greatly saves computation cost. We further optimize the computation process in Equation (15) to save memory cost.

Recoverable Contribution. During pruning, though some channels are pruned, their weights are not set to 0 due to the protection of the mask. When their mask is updated from 0 to 1, they can recover and contribute the accuracy.

5 Experiments

5.1 Datasets and Evaluation Metrics

ADE20K. ADE20K (Zhou et al., 2017; 2019) is a scene parsing dataset containing 25k images in the training set and 2k images in the validation set with 150 label classes.

Cityscapes. Cityscapes (Cordts et al., 2016) is a dataset of urban street scenes from cars collected in 50 cities. It includes 5,000 finely annotated images, in which 2,975 images are used for training, 500 for validation, and 1,525 for testing. We exclude the extra training data with coarse labels throughout this paper. This dataset has 30 label classes, and 19 of them are used for segmentation. The resolution of images is 2048×1024. Cityscapes dataset is an intensively studied benchmark for semantic segmentation, but it is challenging to perform real-time inference on such a high resolution.

Pascal VOC. PASCAL Visual Object Classes (VOC) 2012 (Everingham et al., 2010) is a widely used dataset for semantic segmentation, classification, and object detection tasks. There are 1,464 images for training and 1,449 images for validation. We show the results on Pascal VOC in Appendix D.

Evaluation Metrics. For semantic segmentation evaluation, we use the mean of class-wise intersection-over-union (mIoU) to measure the accuracy performance. We introduce the details of mIoU in Appendix E. For our results, we run our algorithm three times and report the mean and variance of the mIoU performance. For the baseline methods, some methods cost too many resources and we can hardly rerun their experiments. Some methods provide their well-trained models, and we can test with the trained model.

5.2 Experimental Settings

Train Settings. We perform the pruning to search for a suitable width for each CONV layer in the TopFormer model. To enable a larger search space, we adopt the TopFormer architecture and use

Table 1: Comparison of our searched model and prior arts on the ADE20K val dataset. We compare with popular handcraft baselines in the first segment, NAS-based models in the second segment, pruning-based methods in the third segment and lightweight ViT-based models in the fourth segment. We measure the FPS on the Qualcomm Adreno 660 GPU of the Samsung Galaxy S21 mobile phone. Some FPS results are not available due to unsupported operations on the mobile device.

Category	Method	Backbone	Parameters	GMACs	FPS	val mIoU (%)
Human Design	PSPNet (Zhao et al., 2017)	ResNet50-D8	$49.1M$	178.8	0.2	41.1
	DeepLabV3+ (Chen et al., 2018)	MobileNetV2	$15.4M$	25.8	5.7	36.2
	DeepLabV3+ (Chen et al., 2018)	EfficientNet	$17.1M$	26.9	5.9	37.6
	BiSeNetV2 (Yu et al., 2021)	N.A.	$3.34M$	12.4	9.1	25.7
	SFNet (Li et al., 2020b)	ResNet-50	–	75.7	–	42.8
	HRNet-W18-S (Wang et al., 2020)	HRNet-W18-S	$4.0M$	10.2	5.5	31.4
NAS	HR-NAS-A (Ding et al., 2021a)	Searched	$2.5M$	1.4	–	33.2
	HR-NAS-B (Ding et al., 2021a)	Searched	$3.9M$	2.2	–	34.9
	NASViT (Gong et al., 2021)	Searched	–	2.5	–	37.9
	HRViT-b1 (Gu et al., 2021)	Searched	$8.2M$	14.6	–	45.9
Prune	EagleEye (Li et al., 2020a)	N.A.	$3.4M$	1.2	59.2	34.3
	DMCP (Guo et al., 2020a)	N.A.	$3.3M$	1.2	63.8	33.9
	ResPep (Ding et al., 2021b)	N.A.	$3.3M$	1.2	64.9	35.0
	CHEX (Hou et al., 2022)	N.A.	$3.3M$	1.2	64.2	35.2
ViT	Segmenter (Strudel et al., 2021)	Searched	$6.7M$	4.6	4.1	39.9
	MobileViT (Mehta & Rastegari, 2021)	Searched	$3.9M$	2.2	41.8	34.9
	SegFormer-B0 (Xie et al., 2021)	MiT-B0	$3.8M$	8.4	2.6	37.4
	TopFormer-Base (Zhang et al., 2022)	N.A.	$5.1M$	1.8	36.3	37.8
	TopFormer-Small (Zhang et al., 2022)	N.A.	$3.1M$	1.2	54.7	36.1
	TopFormer-Tiny (Zhang et al., 2022)	N.A.	$1.4M$	0.6	82.7	31.8
Ours	**Ours-Base**	Searched	$\mathbf{3.72M}$	**1.8**	**56.5**	**38.9**
	Ours-Small	Searched	$\mathbf{3.25M}$	**1.2**	**75.2**	**37.5**
	Ours-Tiny	Searched	$\mathbf{1.30M}$	**0.7**	**98.0**	**33.5**

a larger per-layer width compared with the TopFormer-Base model. So our unpruned model has 4.1GMACs (with 39.9 mIoU), larger than the 1.8GMACs of the TopFormer-Base model.

We use stochastic gradient descent (SGD) optimizer, and momentum is set to 0.9, and set the batch size to 8 on each GPU. For ADE20K, the initial learning rate is set to 1.2×10^{-4}, and the "poly" learning rate policy is applied. For the Cityscapes dataset, the initial learning rate is set to 3×10^{-4}, and we apply the "poly" learning rate policy. For PASCAL VOC 2012, we set the initial learning rate as 0.01. Learning rate value is determined as $\left(1 - \frac{\text{iter}}{\text{total\_iter}}\right)^{0.9}$ where iter refers to the current iteration number. For the ADE20K dataset, we incorporate data augmentation by resizing with the random ratio between 0.5 and 2.0 as well as random flipping. On the Cityscapes dataset, multiple random scaling $\{0.5, 0.75, 1.0, 2.0\}$ and fixed size cropping of 512×1024 are adopted for data augmentation. We choose the crop size for a better trade-off between mobile capacity and accuracy. We set hyperparameters $\beta = 0.01$ and $\lambda = 0.1$ in the experiments.

Test Settings. Instead of muti-scale testing, we employ single scale testing for a fair comparison. For the ADE20K dataset, we use 512×512 as the input resolution. For the Cityscapes dataset, 512×1024 (rather than 1024×2048) is used as the inference resolution during testing for the following reasons. (i) In practice, we cannot use the resolution 1024×2048 since it causes memory overflow problems on our selected mobile phone. (ii) Besides, since the screens on edge devices such as mobile phones are not very large, the resolution of 512×1024 is good enough to serve on the small screens. (iii) Moreover, we find that this 512×1024 resolution can greatly speedup the inference on the mobile phones without significant accuracy degradation.

Experiment Environments. We train and prune the model using PyTorch 1.9 and CUDA 11.1 on 8 NVIDIA RTX TITAN GPUs. We measure the mobile latency on the GPU of an Samsung Galaxy S21 smartphone, with Qualcomm Snapdragon 888 mobile platform integrated with Qualcomm Kryo 680 Octa-core CPU and a Qualcomm Adreno 660 GPU. Note that for most baseline works, even the reduced resolution (512×1024) can cause an out-of-memory problem on the selected mobile device.

Compiler Framework on Mobile Devices. We need to compile the models before they can be executed on mobile devices. For TopFormer, we adopt the compiler TNN (Contributors, 2019) to report the speed performance, which is also used in the original TopFormer paper. To further improve

Table 2: Our search results on the Cityscapes val dataset. We compare with popular handcraft baselines, NAS-based models, pruning based methods and lightweight ViT-based models.

Category	Method	Backbone	Resolution	#params	GMACs	mIoU%
Human Design	ENet (Paszke et al., 2016)	N.A.	512×1024	$354.9K$	5.9	58.3
	PSPNet (Zhao et al., 2017)	ResNet101	1024×2048	$68.07M$	525.0	78.4
	BiSeNetV2 (Yu et al., 2021)	N.A.	512×1024	$3.34M$	24.6	72.6
	DeepLabV3+ (Chen et al., 2018)	MBv2	512×1024	$2.26M$	9.5	69.0
	STDC1-Seg50 (Fan et al., 2021)	STDC1	512×1024	$12.05M$	31.1	71.9
	STDC2-Seg50 (Fan et al., 2021)	STDC2	512×1024	$16.08M$	44.3	73.4
NAS	Auto-DeepLab-S (Liu et al., 2019)	N.A.	1024×2048	$10.15M$	333.3	79.9
	FasterSeg (Chen et al., 2019)	N.A.	1024×2048	–	28.2	71.5
	DCNAS (Zhang et al., 2021)	N.A.	1024×2048	–	294.6	84.3
Prune	EagleEye(Li et al., 2020a)	N.A.	512×1024	$3.6M$	2.4	69.6
	DMCP (Guo et al., 2020a)	N.A.	512×1024	$3.5M$	2.4	70.3
	ResPep (Ding et al., 2021b)	N.A.	512×1024	$3.5M$	2.4	71.3
	CHEX (Hou et al., 2022)	N.A.	512×1024	$3.4M$	2.4	71.7
ViT	HRViT-b1 (Gu et al., 2021)	N.A.	512×1024	$8.1M$	28.2	81.6
	SegFormer-B0 (Xie et al., 2021)	MiT-B0	512×1024	$3.8M$	17.7	71.9
	TopFormer-Base (Zhang et al., 2022)	N.A.	512×1024	$5.1M$	2.7	70.6
	TopFormer-Tiny (Zhang et al., 2022)	N.A.	512×1024	$1.4M$	1.2	66.1
Ours	**Ours-Base**	Searched	512×1024	$\mathbf{3.7M}$	**3.6**	**74.7**
	Ours-Small	Searched	512×1024	$\mathbf{3.3M}$	**2.4**	**73.6**
	Ours-Tiny	Searched	512×1024	$\mathbf{1.3M}$	**1.4**	**71.5**

the inference speed, we adopt several compiler optimization methods and develop an advanced compiler framework to compile our derived models and test their speed on mobile devices. We show the details about our compiler optimization in Appendix F.

5.3 Experimental Results and Analysis

Segmentation Performance. Based on the TopFormer model, we obtain three models (Ours-Base, Ours-Small, and Ours-Tiny) with different computations. We show the comparison results on ADE20K in Table 1 and Cityscapes in Table 2. (i) For ADE20K, we can observe that the human-designed segmentation models and NAS-based models usually consume many computations (such as DeepLabV3+ with 25.8GMACs) in terms of MACs (multiply–accumulate operations). They can hardly achieve real-time inference on edge devices. Some NAS-based models with a small number of computations are not able to achieve high mIoU (such as HR-NAS-A with 33.2 mIoU). (ii) For the comparison with other pruning methods, we start from the same dense model and set the target GMACs to the same number (1.2GMACs) to make a fair comparison. We can see that our small model can achieve higher mIoU given the same GMACs. (iii) Compared with the ViT-based TopFormer, our models can achieve higher mIoUs with non-trivial improvements under the same computation budget (such as Ours-Small 37.5 mIoU v.s. 36.1 of TopFormer-Small under the same 1.2GMACs). Our models can achieve a faster inference speed (FPS higher than 50) on mobile phones compared with TopFormer models. We show the comparison with the baselines in terms of mIoU and FPS in Figure 1. We can achieve a better trade-off between mIoU and FPS.

(i) On the Cityscapes dataset, compared with handcrafted CNN-based segmentation models, including BiSeNetV2 and STDC, our searched base model greatly reduces the GMACs (such as Ours-Base 3.6GMACs v.s. 24.6GMACs of BiSeNetV2) and achieves non-trivial better accuracy (Ours-Base 74.7 mIoU v.s. 72.6 mIoU of BiSeNetV2), as shown in Table 2. (ii) Compared with NAS-based methods such as FasterSeg, our models can achieve higher mIoU with much smaller computation costs. Other NAS methods consume too many computations (e.g., DCNAS with 300GMACs) to be deployed on edge devices. (iii) Compared with other pruning baselines, under the same computation budgets (2.4GMACs), our small model can achieve higher mIoU. (iv) Compared with transformer-based methods such as SegFormer-B0, similarly, our models achieve higher mIoU with less computations. Our small and tiny models have similar computations compared with TopFormer-Base and TopFormer-Tiny. But we can achieve higher mIoU.

Speed Performance on Mobile Devices. Our base model can achieve 56.5 FPS on the mobile device (Samsung Galaxy S21), which implements real-time execution with competitive segmentation

performance, as shown in Table 1. Other baseline methods except TopFormer can hardly achieve real-time segmentation on edge devices, usually with FPS lower than 10. Our faster inference speed than TopFormer is achieved with the compiler optimization techniques, detailed in Appendix F.

Search Overhead. We show the search cost in Table 3. We only show the cost of our small model since our base, small and tiny models have similar search costs. It takes approximately 1.3 GPU days, which is smaller than the search cost of most other NAS-based segmentation methods. Our method can efficiently search out a compact model with fewer computations and better segmentation performance. The low search cost is achieved by our pruning parameterization framework. Based on the soft mask and the low-cost thresholding and STE method, we are able to directly train the model weights and the pruning parameters. During the training, the pruned channels are zeroed out by the binary mask, and we do not need additional overhead for fine-tuning. Besides, our bi-level optimization can efficiently address the second-order derivatives with low computation complexity.

Visualization Comparison. We show the visualization comparison of our base model and other baseline methods in Appendix G. Our method can achieve better segmentation performance.

Results on Other Datasets and Model Architectures. We show the results of our method on the Pascal VOC dataset in Appendix D. Besides, our method is general and can be applied to other model structures. We show the results for DeepLabV3+ (Chen et al., 2018) on Cityscapes in Appendix H.

Method	GPU Days	GMACs	mIoU
Auto-DeepLab (Liu et al., 2019)	3	695.0	82.1
GAS (Lin et al., 2020)	6.7	-	73.5
FasterSeg (Chen et al., 2019)	2	28.2	71.5
Fast-NAS (Nekrasov et al., 2019)	8	435.7	78.9
SparseMask (Wu et al., 2019)	4.2	36.4	68.6
DCNAS (Zhang et al., 2021)	5.6	94.6	84.3
LDP (Huynh et al., 2022)	4.3	–	75.8
Without implicit gradients	1.1	2.4	71.9
Ours-Small	1.3	2.4	73.6

Table 3: Comparison of search cost on the Cityscapes validation dataset.

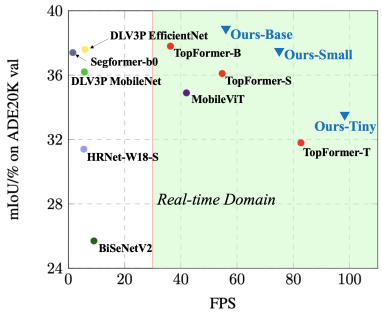


Figure 1: Comparison of accuracy versus FPS on ADE20K val dataset.

5.4 Ablation Study

In our bi-level optimization, we incorporate implicit gradients in Equation (11). To demonstrate the advantages with implicit gradients, we consider the case without implicit gradients which omits the second term in Equation (11) with just $\frac{d\mathcal{L}_m(\boldsymbol{w}^*,\boldsymbol{s})}{d\boldsymbol{s}} = \nabla_{\boldsymbol{s}}\mathcal{L}_m(\boldsymbol{w}^*,\boldsymbol{s})$. We compare the performance of the solution without implicit gradients and our bi-level solution with implicit gradients on Cityscapes in Table 3. We can observe that our solution with implicit gradients has a search cost slightly higher than the solution without implicit gradients, demonstrating that our first-order solution for the second-order derivatives in Equation (12) can effectively save computation cost. For mIoU, our method can achieve higher mIoU with non-trivial improvements, demonstrating that incorporating implicit gradients can effectively boost the performance. We show the results with various β and λ in Appendix I. To compare with baselines under certain computations, we mainly show the results of our three sparse models (Ours-Base, Ours-Small, and Ours-Tiny). We show the results of more models under other computations in Appendix J.

6 Conclusion

We propose pruning parameterization with the thresholding and STE methods to build a pruning framework. Based on the framework, we formulate the problem and propose a bi-level optimization method with the implicit gradients. Our experimental results demonstrate that we can achieve the highest mIoU under the same computation constraint on various datasets. Specifically, we can achieve 38.9 mIoU on the ADE20K with a real-time inference speed of 56.5 FPS on the Samsung S21.

ETHICS STATEMENT

As we try to prune the model to achieve better segmentation performance and faster inference speed, currently we do not think of any potential ethics concern. We use open datasets with images of street scenes. There might be human faces shown in these images. But it is hard to clearly recognize the human faces as it is only a very small region in the street scene images.

REPRODUCIBILITY STATEMENT

For the experiments, in Section 5.1, we discuss the details about datasets and evaluation metrics. In Section 5.2, we show the details about the training and testing including the hyper-parameter setting, image resolutions, experiment devices, mobile phones and so on.

For the theory part, we show the detailed step-by-step derivation in Section 4.1. For the key equations such as Equation (2), (9) and (12), we show their proof in Appendix A, B, and C.

REFERENCES

Anjali Balagopal, Howard Morgan, Michael Dohopoloski, Ramsey Timmerman, Jie Shan, Daniel F. Heitjan, Wei Liu, Dan Nguyen, Raquibul Hannan, Aurelie Garant, Neil Desai, and Steve Jiang. Psa-net: Deep learning based physician style-aware segmentation network for post-operative prostate cancer clinical target volume, 2021. 16

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432. 3, 4

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 3

Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 801–818, 2018. 2, 7, 8, 9, 16, 17, 18

Wuyang Chen, Xinyu Gong, Xianming Liu, Qian Zhang, Yuan Li, and Zhangyang Wang. Fasterseg: Searching for faster real-time semantic segmentation. *arXiv preprint arXiv:1912.10917*, 2019. 2, 8, 9

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive darts: Bridging the optimization gap for nas in the wild. *International Journal of Computer Vision*, pp. 1–18, 2020. 5

Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Xiaoyi Dong, Lu Yuan, and Zicheng Liu. Mobile-former: Bridging mobilenet and transformer. *arXiv preprint arXiv:2108.05895*, 2021. 3

Bowen Cheng, Ishan Misra, Alexander G Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. *arXiv preprint arXiv:2112.01527*, 2021a. 1, 3

Bowen Cheng, Alex Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. *Advances in Neural Information Processing Systems*, 34, 2021b. 1, 3

Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019. 3

TNN Contributors. TNN: A high-performance, lightweight neural network inference framework. https://github.com/Tencent/TNN, 2019. 7

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 6

Mingyu Ding, Xiaochen Lian, Linjie Yang, Peng Wang, Xiaojie Jin, Zhiwu Lu, and Ping Luo. Hr-nas: Searching efficient high-resolution neural architectures with lightweight transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021a. 2, 7

Xiaohan Ding, Tianxiang Hao, Jianchao Tan, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. Resrep: Lossless cnn pruning via decoupling remembering and forgetting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4510–4520, 2021b. 7, 8

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021. 1, 2

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*, 2018. 3

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010. 6

Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. On the convergence theory of gradient-based model-agnostic meta-learning algorithms. In *International Conference on Artificial Intelligence and Statistics*, pp. 1082–1092. PMLR, 2020. 5

Mingyuan Fan, Shenqi Lai, Junshi Huang, Xiaoming Wei, Zhenhua Chai, Junfeng Luo, and Xiaolin Wei. Rethinking bisenet for real-time semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9716–9725, 2021. 1, 2, 8

Jun Fu, Jing Liu, Haijie Tian, Yong Li, Yongjun Bao, Zhiwei Fang, and Hanqing Lu. Dual attention network for scene segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3146–3154, 2019. 2

Shangqian Gao, Feihu Huang, Jian Pei, and Heng Huang. Discrete model compression with resource constraint for deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 4

Chengyue Gong, Dilin Wang, Meng Li, Xinlei Chen, Zhicheng Yan, Yuandong Tian, Vikas Chandra, et al. Nasvit: Neural architecture search for efficient vision transformers with gradient conflict aware supernet training. In *International Conference on Learning Representations*, 2021. 2, 7

Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *CoRR*, abs/1607.05447, 2016. URL http://arxiv.org/abs/1607.05447. 4

Jiaqi Gu, Hyoukjun Kwon, Dilin Wang, Wei Ye, Meng Li, Yu-Hsin Chen, Liangzhen Lai, Vikas Chandra, and David Z Pan. Hrvit: Multi-scale high-resolution vision transformer. *arXiv preprint arXiv:2111.01236*, 2021. 7, 8

Yushuo Guan, Ning Liu, Pengyu Zhao, Zhengping Che, Kaigui Bian, Yanzhi Wang, and Jian Tang. Dais: Automatic channel pruning via differentiable annealing indicator search. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2022. doi: 10.1109/TNNLS.2022.3161284. 1, 4

Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. Dmcp: Differentiable markov channel pruning for neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1539–1547, 2020a. 4, 7, 8

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, pp. 544–560. Springer, 2020b. 3

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 1389–1397, 2017. 4

Zejiang Hou, Minghai Qin, Fei Sun, Xiaolong Ma, Kun Yuan, Yi Xu, Yen-Kuang Chen, Rong Jin, Yuan Xie, and Sun-Yuan Kung. Chex: Channel exploration for cnn model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12287–12298, 2022. 7, 8

Junjie Huang, Zheng Zhu, and Guan Huang. Multi-stage hrnet: multiple stage high-resolution network for human pose estimation. *arXiv preprint arXiv:1910.05901*, 2019. 2

Lam Huynh, Esa Rahtu, Jiri Matas, and Janne Heikkila. Fast neural architecture search for lightweight dense prediction networks. *arXiv preprint arXiv:2203.01994*, 2022. 9

Minsoo Kang and Bohyung Han. Operation-aware soft channel pruning using differentiable masks. In *International Conference on Machine Learning*, pp. 5122–5131. PMLR, 2020. 4

Jaedeok Kim, Chiyoun Park, Hyun-Joo Jung, and Yoonsuck Choe. Plug-in, trainable gate for streamlining arbitrary neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):4452–4459, Apr. 2020. doi: 10.1609/aaai.v34i04.5872. URL https://ojs.aaai.org/index.php/AAAI/article/view/5872. 4

Bailin Li, Bowen Wu, Jiang Su, and Guangrun Wang. Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In *European conference on computer vision*, pp. 639–654. Springer, 2020a. 7, 8

Tuanhui Li, Baoyuan Wu, Yujiu Yang, Yanbo Fan, Yong Zhang, and Wei Liu. Compressing convolutional neural networks via factorized convolutional filters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3977–3986, 2019a. 3

Xia Li, Zhisheng Zhong, Jianlong Wu, Yibo Yang, Zhouchen Lin, and Hong Liu. Expectation-maximization attention networks for semantic segmentation, 2019b. 16

Xiangtai Li, Ansheng You, Zhen Zhu, Houlong Zhao, Maoke Yang, Kuiyuan Yang, Shaohua Tan, and Yunhai Tong. Semantic flow for fast and accurate scene parsing. In *European Conference on Computer Vision*, pp. 775–793. Springer, 2020b. 1, 2, 7

Xin Li, Yiming Zhou, Zheng Pan, and Jiashi Feng. Partial order pruning: for best speed/accuracy trade-off in neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9145–9153, 2019c. 2

Peiwen Lin, Peng Sun, Guangliang Cheng, Sirui Xie, Xi Li, and Jianping Shi. Graph-guided architecture search for real-time semantic segmentation, 2020. 9

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 82–92, 2019. 2, 8, 9

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 3

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pp. 2736–2744, 2017. 1, 4

Sachin Mehta and Mohammad Rastegari. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021. 3, 7, 16

Vladimir Nekrasov, Hao Chen, Chunhua Shen, and Ian Reid. Fast neural architecture search of compact semantic segmentation models via auxiliary cells, 2019. 9

Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation, 2016. 2, 8

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018. 3

Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in neural information processing systems*, 32, 2019. 5

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019. 3

Walter Rudin et al. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1976. 5, 15

Kegan GG Samuel and Marshall F Tappen. Learning optimized map estimates in continuously-valued mrf models. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 477–484. IEEE, 2009. 5

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018. 2

Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7262–7272, 2021. 7

Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 16

Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3349–3364, 2020. 7

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems (NeurIPS)*, pp. 2074–2082, 2016. 3

Huikai Wu, Junge Zhang, and Kaiqi Huang. Sparsemask: Differentiable connectivity learning for dense image prediction, 2019. 9

Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34, 2021. 7, 8

Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 32, 2019. 4

Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018. 2

Changqian Yu, Changxin Gao, Jingbo Wang, Gang Yu, Chunhua Shen, and Nong Sang. Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation. *International Journal of Computer Vision*, pp. 1–18, 2021. 1, 2, 7, 8

Wenqiang Zhang, Zilong Huang, Guozhong Luo, Tao Chen, Xinggang Wang, Wenyu Liu, Gang Yu, and Chunhua Shen. Topformer: Token pyramid transformer for mobile semantic segmentation. *arXiv preprint arXiv:2204.05525*, 2022. 1, 3, 7, 8, 16

Xiong Zhang, Hongmin Xu, Hong Mo, Jianchao Tan, Cheng Yang, Lei Wang, and Wenqi Ren. Dcnas: Densely connected neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13956–13967, 2021. 2, 8, 9

Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2881–2890, 2017. 2, 7, 8

Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip HS Torr, et al. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6881–6890, 2021. 1, 3

Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 6

Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127(3):302–321, 2019. 6

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017. 3

Appendix

A Binary Mask

Here we show that by using the binary mask $\boldsymbol{b}_l$ to perform depth-wise CONV, the corresponding output channels are pruned.

Let $\boldsymbol{b}_l = \{0\}^{o_0 \times 1 \times 1 \times 1} \oplus \{1\}^{o_1 \times 1 \times 1 \times 1}$, where o_0 and o_1 denote the number of zeros and ones in $\boldsymbol{b}_l$, respectively, with $o_0 + o_1 = o$, and $\oplus$ refers to channel-wise concatenation. Thus we have

$$\begin{aligned} \boldsymbol{a}_l &= \boldsymbol{b}_l^{o \times 1 \times 1 \times 1} \odot (\boldsymbol{w}_l^{o \times i \times k \times k} \odot \boldsymbol{a}_{l-1}) \\ &= (\{0\}^{o_0 \times 1 \times 1 \times 1} \cdot \boldsymbol{w}_l^{o_0 \times i \times k \times k} \odot \boldsymbol{a}_{l-1}) \oplus (\{1\}^{o_1 \times 1 \times 1 \times 1} \cdot \boldsymbol{w}_l^{o_1 \times i \times k \times k} \odot \boldsymbol{a}_{l-1}) \\ &= \boldsymbol{w}_l^{o_1 \times i \times k \times k} \odot \boldsymbol{a}_{l-1}. \end{aligned} \tag{16}$$

We can see that after using the binary mask $\boldsymbol{b}_l$ to perform depth-wise CONV with $\boldsymbol{w}_l \odot \boldsymbol{a}_{l-1}$, the output channels corresponding to the zero elements in $\boldsymbol{b}_l$ are pruned, and only channels corresponding to the non-zero elements in $\boldsymbol{b}_l$ are left.

B Steps to Obtain Implicit Gradients

We show how to obtain $\frac{d\boldsymbol{w}^*}{d\boldsymbol{s}}$. Since $\boldsymbol{w}^*$ is optimal, we have

$$\nabla_{\boldsymbol{w}} g(\boldsymbol{w}^*, \boldsymbol{s}) = 0, \tag{17}$$

where $g(\boldsymbol{w}, \boldsymbol{s}) = \mathcal{L}(\boldsymbol{w}, \boldsymbol{s}) + \frac{1}{2\lambda}\boldsymbol{w}^T\boldsymbol{w}$. By implicit function theorem (Rudin et al., 1976), we have

$$\frac{d\nabla_{\boldsymbol{w}} g(\boldsymbol{w}^*, \boldsymbol{s})}{d\boldsymbol{s}} = 0, \tag{18}$$

Applying the chain rule, we can obtain

$$\nabla^2_{\boldsymbol{s}\boldsymbol{w}} g(\boldsymbol{w}^*, \boldsymbol{s}) + \frac{d\boldsymbol{w}^*}{d\boldsymbol{s}} \nabla^2_{\boldsymbol{w}} g(\boldsymbol{w}^*, \boldsymbol{s}) = 0. \tag{19}$$

$\frac{d\boldsymbol{w}^*}{d\boldsymbol{s}}$ can be obtained through

$$\frac{d\boldsymbol{w}^*}{d\boldsymbol{s}} = -\nabla^2_{\boldsymbol{s}\boldsymbol{w}} g(\boldsymbol{w}^*, \boldsymbol{s}) \nabla^2_{\boldsymbol{w}} g(\boldsymbol{w}^*, \boldsymbol{s})^{-1}. \tag{20}$$

C Computation of Second-Order Information

We show how to obtain the second-order derivatives. Using the chain rule, we can obtain that

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{s}) = \text{diag}(\boldsymbol{s}) \nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b) = \boldsymbol{s} \cdot \nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b), \tag{21}$$

$$\nabla_{\boldsymbol{s}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{s}) = \text{diag}(\boldsymbol{w}^*) \nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b) = \boldsymbol{w}^* \cdot \nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b) \tag{22}$$

Thus

$$\begin{aligned} \nabla_{\boldsymbol{s}\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{s}) &= \nabla_{\boldsymbol{s}} [\boldsymbol{s} \cdot \nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b)] && (23) \\ &= \text{diag}(\nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b)) + \text{diag}(\boldsymbol{s}) [\nabla_{\boldsymbol{s}} (\nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b))] && (24) \\ &= \text{diag}(\nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b)) + \text{diag}(\boldsymbol{s}) [\nabla_{\boldsymbol{w}^*} (\nabla^2_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b))] && (25) \\ &= \text{diag}(\nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b)) && (26) \end{aligned}$$

where the last equality holds due to the Hessian-free assumption.

D Results on PASCAL VOC

For the PASCAL VOC 2012 test dataset, the input images are randomly cropped to 512×512. We show the results on PASCAL VOC 2012 test in Table A1. We can observe that handcrafted CNN-based methods usually require more computational cost (such as DeepLabV3+ with 5.7 GMACs for MobileNetV2 backbone and 37.8 GMACs for ResNet50 backbone). Compared with ViT-based TopFormer, our method could achieve better mIoU under the same computational cost (such as Ours-Small 73.4% mIoU v.s. 69.8% mIoU of TopFormer-S for 1.2 GMACs).

Table A1: Results on the PASCAL VOC 2012 test dataset. We compare our results with popular CNN-based models and lightweight ViT-based models.

Method	#params	GMACs	mIoU%	FPS
EfficientNet-B7 (Tan & Le, 2019)	$66.0M$	194.0	85.2	0.1
EMANet (Li et al., 2019b)	$10.0M$	43.1	80.1	2.5
PSANet (Balagopal et al., 2021)	$18.5M$	56.3	78.5	1.4
DeepLabV3+ R101 (Chen et al., 2018)	$43.9M$	58.5	77.4	2.2
DeepLabV3+ R50 (Chen et al., 2018)	$24.9M$	37.8	76.3	3.1
DeepLabV3+ MBv2 (Chen et al., 2018)	$2.3M$	5.7	70.5	5.1
TopFormer-B (Zhang et al., 2022)	$5.1M$	1.8	71.0	36.8
TopFormer-S (Zhang et al., 2022)	$3.1M$	1.2	69.8	55.2
TopFormer-T (Zhang et al., 2022)	$1.4M$	0.6	65.7	81.5
MobileViT-XXS (Mehta & Rastegari, 2021)	$1.9M$	1.7	73.6	43.8
Ours-Base	$3.7M$	1.8	74.3 ± 0.29	56.8
Ours-Small	$3.3M$	1.2	73.4 ± 0.34	75.0
Ours-Tiny	$1.3M$	0.7	70.5 ± 0.38	97.6

E Details of mIoU

The computation of mIoU is shown below,

$$mIoU = \frac{1}{n} \sum_{i}^{n} \frac{\sum P^i_{overlap}}{\sum P^i_{union}}, \tag{27}$$

where n is the class number (e.g., 19 for Cityscapes), and P_i refers to pixels that are assigned to a specific class label i.

F Details of Compiler Optimization

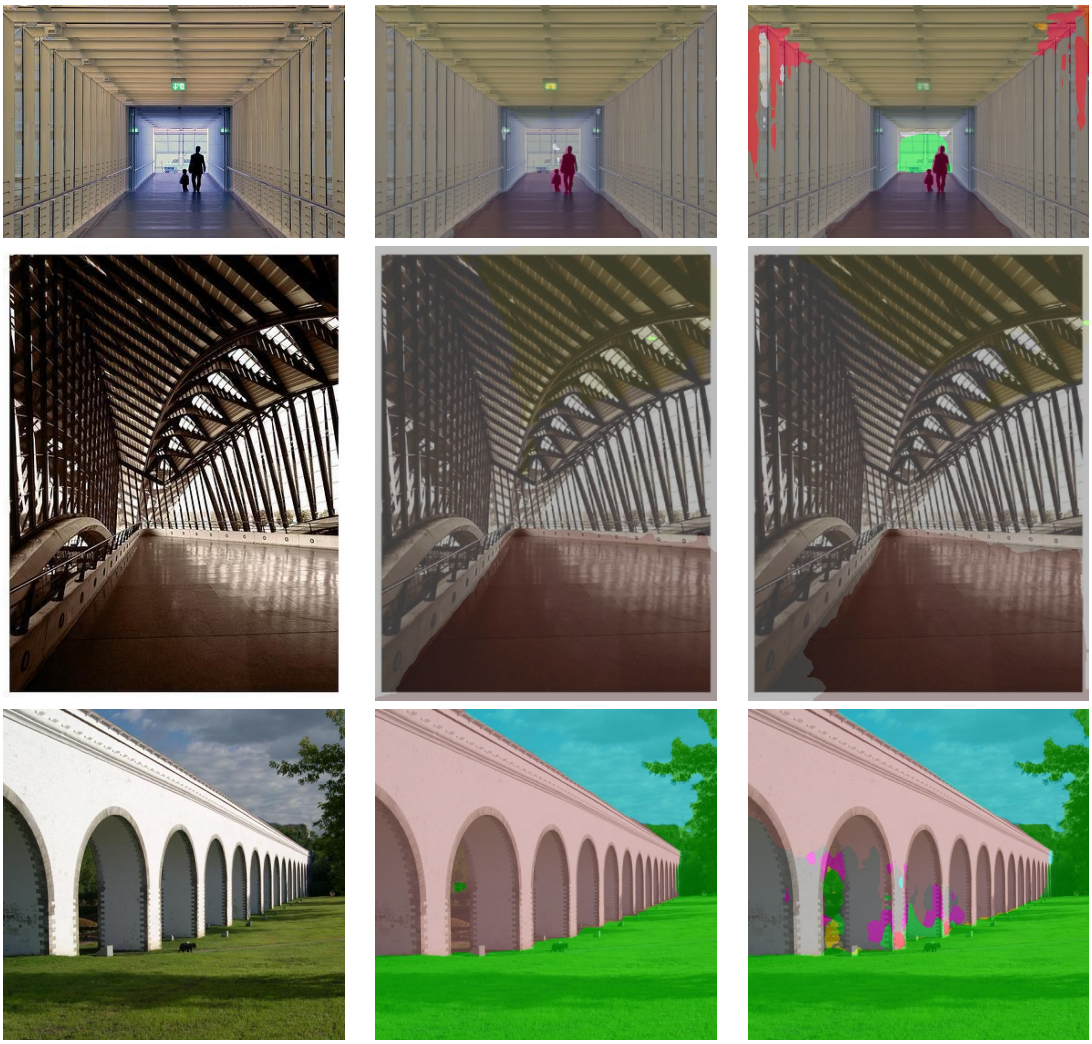
Compiler optimization can support various pruning ratios. The compiler optimization consists of the following components in detail.

Sparse Model Storage. To further improve data locality, compared with the well-known CSR, a more compact format is adopted to store the sparse model weights. We avoid storing zero weights of the model to achieve a high compression rate. We remove redundant indices from the structured pruning. The sparse model storage can save the scarce memory bandwidth of mobile devices.

Layer Fusion. Layer fusion is commonly adopted in compiler optimization to fuse the computation operators in the computation graph. With the help of layer fusion, we can avoid saving the parameters of fused operators and their intermediate computation results. The operator number is also reduced. For layer fusion, based on the computation laws such as associative property and distributive property, we identify some operator combinations which are available for fusion. The basic rule is to check whether the fusion can enlarge the overall computation for CPU/GPU utilization improvement and reduce the memory access for memory efficiency. For example, a combination of the Convolution layer (or Depthwise Convolution layer) and its following BatchNorm layer can be fused into one layer to reduce the data movement and access with higher instruction level parallelism.

Matrix Reorder. There are some well-known challenges for sparse matrix multiplications, such as the heavy load imbalance among each thread and irregular memory accesses. To deal with these challenges, a matrix reorder method is adopted to leverage the structure information from the structured pruning. For example, for the column pruning to remove the whole columns, there is a certain degree of regularity as the rest weights are stored in unpruned columns. Thus, matrix reorder rearranges the rows with the same or similar patterns together, i.e., reorders the rows. Then, matrix reorder makes the weights in the column direction (e.g., kernels in CNN) more compact.

Parameter Auto-Tuning. During compiler optimization, there are many parameters, such as data placement on GPU memory, loop unrolling factors, matrix tiling sizes, etc. The compiler adopts an



(a) Input (b) Ours-Base (c) TopFormer-Base

Figure A1: Visualization results on samples of ADE20K validation dataset.

auto-tuning method to find the best configuration of the parameters. Specifically, a genetic algorithm is used to search the parameter space. Besides, we can use a larger population number in each generation to improve the exploration parallelism.

G VISUALIZATION COMPARISON

We visualized the inference results of TopFormer-Base and Ours-Base on the ADE20K validation dataset in Figure A1 and the Cityscapes validation dataset in Figure A2. Ours-Base model can achieve better visualization performance than the TopFormer-Base model.

H RESULTS WITH DEEPLABV3+ ON CITYSCAPES

We show our experiment results for the DeepLabV3+ model (Chen et al., 2018) on the Cityscapes validation dataset in Table A2. The backbone we used is MobileNetV2. After our search, we get two models. Both models have fewer parameters and lower computational costs while achieving better mIoU. The latency on the mobile phone is also less than the original DeepLabV3+ model.

(a) Input (b) Ours-Base (c) TopFormer-Base

Figure A2: Visualization results on samples of Cityscapes validation dataset.

Table A2: Our searched results on DeepLabV3+ with MobileNetV2 backbone. Dataset is Cityscapes val. FPS is measured on the Qualcomm Adreno 660 GPU of the Samsung S21 mobile phone.

Method	Backbone	Resolution	#params	GMACs	mIoU%	FPS
DeepLabV3+ (Chen et al., 2018)	MBv2	512×1024	$2.26M$	9.5	69.0	16.4
Ours-Base	MBv2-Searched	512×1024	$1.56M$	**8.1**	**70.6**	**19.2**
Ours-Small	MBv2-Searched	512×1024	$686.4K$	**5.2**	**70.1**	**24.6**

I Results with Different Hyperparameters

We experiment with different values of β and show the results in Table A3. The target MACs is 2.4G. We choose $\beta = 0.01$ as it can achieve the best performance. If β is too small, it can hardly obtain the target MACs requirement. We show the results of different λ in Table A4.

J More Pruned Models

We conduct additional experiments to obtain models with other compression rates based on our unpruned model. As shown in Table A5 and A6, we can see that usually, the mIoU drops as we prune more parameters, and when the parameter counts are above 60%, the mIoU can be kept above 76.1 and 39.6 for Cityscapes and ADE20K dataset, respectively, which are close to the original mIoU of the unpruned model with 76.5 and 39.9 mIoU. In the extreme case where the parameter counts are only 9% of the unpruned model, our mIoU is still higher than the SOTA TopFormer baseline with fewer parameters and computations.

Table A3: Results on Cityscapes with different β.

β	0.001	0.01	0.1	1.0
mIoU	73.1	73.6	72.8	71.2

Table A4: Results on Cityscapes with different λ.

λ	0.01	0.1	1.0
mIoU	73.1	73.6	70.8

Table A5: Comparison of our searched models and TopFormer-B on the Cityscapes dataset.

	Supernet	Ours					TopFormer-B
# Params	$10.34M$	$6.2M$	$3.7M$	$3.3M$	$1.3M$	$0.9M$	$5.1M$
% Params	100%	60%	36%	32%	13%	9%	–
GMACs	8.2	6.2	3.6	2.4	1.4	1.0	2.7
mIoU	76.5	76.1	74.7	73.6	71.5	70.7	70.6

Table A6: Comparison of our searched models and TopFormer-T on ADE20K dataset.

	Supernet	Ours					TopFormer-B
# Params	$10.34M$	$6.2M$	$3.7M$	$3.3M$	$1.3M$	$0.9M$	$1.4M$
% Params	100%	60%	36%	32%	13%	9%	–
GMACs	4.1	3.1	1.8	1.2	0.7	0.5	0.6
mIoU	39.9	39.6	38.9	37.5	33.5	32.0	31.8