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

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

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 [54], 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

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

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

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

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

$$\nabla_{\boldsymbol{sw}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{s}) = \nabla_{\boldsymbol{s}} [\boldsymbol{s} \cdot \nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b)] \tag{23}$$

$$= \text{diag}(\nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b)) + \text{diag}(\boldsymbol{s}) [\nabla_{\boldsymbol{s}} (\nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b))] \tag{24}$$

$$= \text{diag}(\nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b)) + \text{diag}(\boldsymbol{s}) [\nabla_{\boldsymbol{w}^*} (\nabla^2_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b))] \tag{25}$$

$$= \text{diag}(\nabla_{\boldsymbol{w}_b} \mathcal{L}(\boldsymbol{w}_b)) \tag{26}$$

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

D. 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.

E. 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 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.

F. 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.

G. Results with Different hyperparameters

We experiment with different values of β and show the results in Table A1. 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 A2.

H. More Pruned models

We conduct additional experiments to obtain models with other compression rates based on our unpruned model. As shown in Table A3 and A4, 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 A1. 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 A2. Results on Cityscapes with different λ.

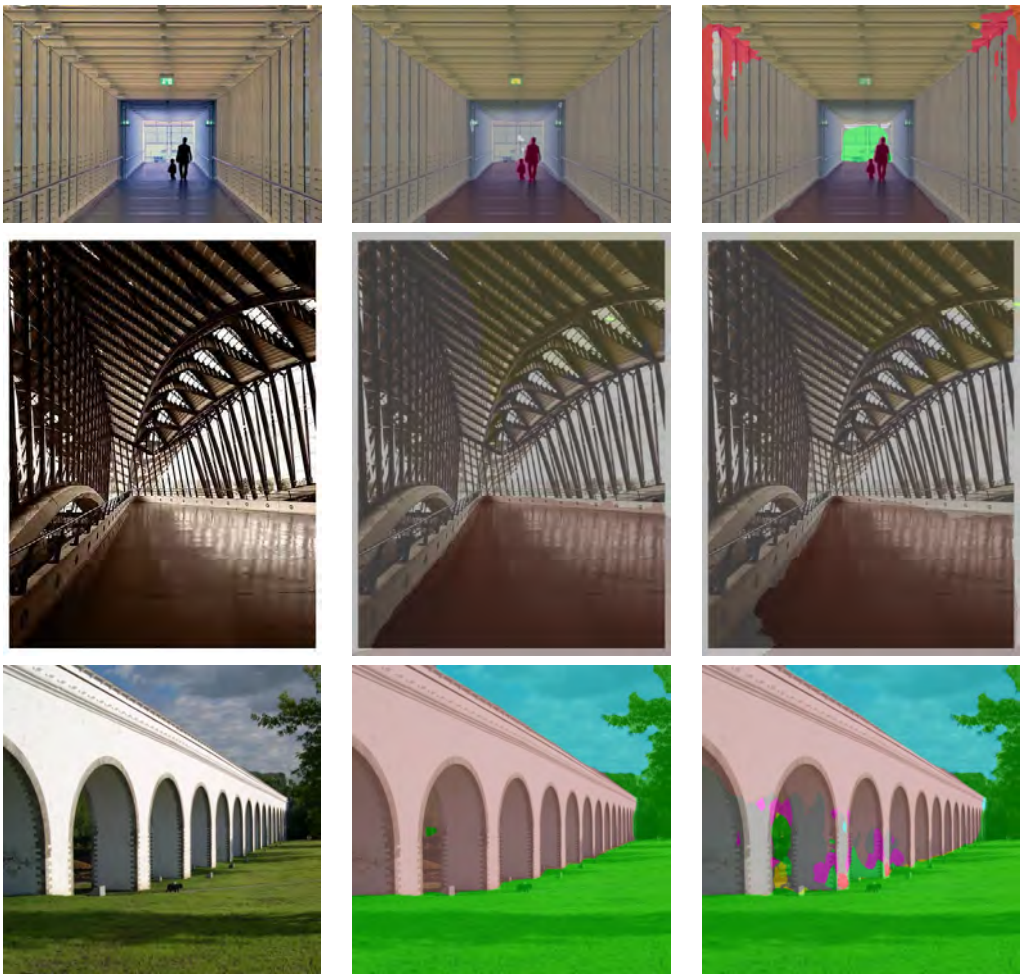
λ	0.01	0.1	1.0
mIoU	73.1	73.6	70.8

Table A3. 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 A4. 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



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

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

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

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