DYNAMICALLY PRUNING SEGFORMER FOR EFFICIENT SEMANTIC SEGMENTATION

Haoli Bai [ξ†], *Hongda Mao* [ξ], *Dinesh Nair* [ξ]

[ξ]Amazon Halo Health & Wellness, Sunnyvale, USA
[†]The Chinese University of Hong Kong, Hong Kong, China

ABSTRACT

As one of the successful Transformer-based models in computer vision tasks, SegFormer demonstrates superior performance in semantic segmentation. Nevertheless, the high computational cost greatly challenges the deployment of SegFormer on edge devices. In this paper, we seek to design a lightweight SegFormer for efficient semantic segmentation. Based on the observation that neurons in SegFormer layers exhibit large variances across different images, we propose a dynamic gated linear layer, which prunes the most uninformative set of neurons based on the input instance. To improve the dynamically pruned SegFormer, we also introduce two-stage knowledge distillation to transfer the knowledge within the original teacher to the pruned student network. Experimental results show that our method can significantly reduce the computation overhead of SegFormer without an apparent performance drop. For instance, we can achieve 36.9% mIoU with only 3.3G FLOPs on ADE20K, saving more than 60% computation with the drop of only 0.5% in mIoU.

***Index Terms*—** Dynamic Pruning, SegFormer, Semantic Segmentation

1. INTRODUCTION

The recent advances of vision transformers (ViT) [1] have inspired a new series of models in computer vision tasks [2, 3, 4, 5, 6]. Among ViT variants, SegFormer [6] extracts hierarchical representations from the input image with the transformer architecture, which shows superior performance in semantic segmentation over the past convolutional neural networks (CNNs) [7, 8, 9, 10, 11].

Despite the empirical success, the SegFormer architecture still suffers high computational cost that challenges the deployment on low-power devices, such as mobile phones or wristbands. The challenges are mainly from the increasing width of fully connected layers to extract high-level visual features in the Mix Transformer (MiT) [6]. Different from previous CNN-based hierarchical architectures [12, 13, 14], the wide and dense linear layers in the MiT encoder inevitably increase the computation overhead.

In this paper, we aim to design efficient SegFormer architecture by dynamically pruning the redundant neurons in the MiT encoder. We find that different neurons in a MiT layer exhibit *large variances* across various input instances. Motivated by the observation, it would be promising if we can identify the best set of neurons to explain the current input, while pruning away the rest uninformative ones. Towards that end, we propose a *dynamic gated linear layer* which computes an instance-wise gate via a lightweight gate predictor. The gate thus selects the best subset of neurons for current input and reduces the computation in the matrix multiplication. Furthermore, we combine *two-stage knowledge distillation* [15] to transfer the knowledge from the original SegFormer (i.e., the teacher) to the dynamically pruned one (i.e., the student). The two-stage distillation minimizes the discrepancy of MiT encoder representations and output logits between the teacher and student model respectively.

Empirical results on ADE20K and CityScape benchmark dataset demonstrate the superiority of our method w.r.t. both performance and efficiency over a number of previous counterparts. For example, the dynamically pruned SegFormer can achieve 36.9% mIoU with only 3.3G FLOPs, which is far smaller than the original 8.4G FLOPs with 37.4% mIoU.

2. PRELIMINARIES

2.1. The SegFormer Architecture

SegFormer aims to extract hierarchical feature representations at multiple scales for semantic segmentation. SegFormer has a Mix Transformer (MiT) encoder followed by a dense MLP decoder, as shown in the left side of Figure 1. The encoder of SegFormer (i.e., MiT) has four different stages. At stage-i of MiT, the input feature map $\bar{\mathbf{X}}_i \in \mathbb{R}^{C_i \times H \times W}$ is transformed to a patch embedding $\mathbf{X}_i = \text{PatchMerge}(\bar{\mathbf{X}}_i) \in \mathbb{R}^{N \times C_i}$, where N is the sequence length, and C_i is the hidden dimension for stage-i. The efficient multi-head attention (MHA) of each transformer layer then computes the query $\mathbf{Q} = \text{FC}(\mathbf{X}_i) \in \mathbb{R}^{N \times C_i}$, key $\mathbf{K} = \text{FC}(\mathbf{X}_i) \in \mathbb{R}^{\bar{N} \times C_i}$ and value $\mathbf{V} = \text{FC}(\mathbf{X}_i) \in \mathbb{R}^{\bar{N} \times C_i}$, where $\text{FC}(\cdot)$ is the linear layer. Note that both $\mathbf{K}$ and $\mathbf{V}$ have smaller sequence lengths $\bar{N} = N/R$ so as to reduce the time complexity of self-attention to $O(N^2/R)$. The output of

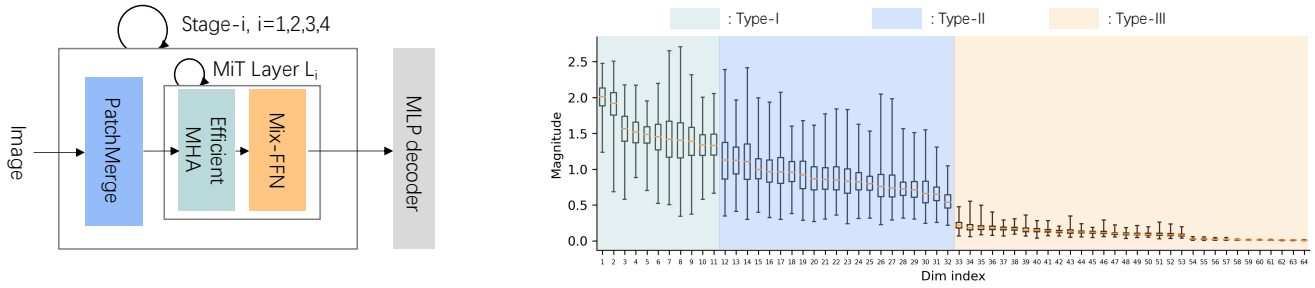

Fig. 1: (**Left**) shows the architecture of SegFormer, which is composed of four stages. Each stage has multiple MiT layers, where each layer has an MHA module followed by the mix FFN module. The decoder adopts a MLP architecture fo pixel-wise prediction. (**Right**) illustrates the neuron variance on the key layer of stage-1.1 in MiT-B0 during inference on ADE20K. The orange lines denote the median, and the rectangles being the lower and upper quartile. The black lines represent the overall range of the neuron.

MHA $\mathbf{X}_{mha}$ thus can be obtained by

$$\mathbf{A} = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{\bar{N}}})\mathbf{V} \in \mathbb{R}^{N\times C_i}, \quad (1)$$

$$\mathbf{X}_{mha} = \text{LN}\big(\hat{\mathbf{X}} + \text{FC}(\mathbf{A})\big) \in \mathbb{R}^{N\times C_i}, \quad (2)$$

where $\text{LN}(\cdot)$ denotes the layer-normalization layer. Then the mix feed-forward network (Mix-FFN) transforms $\mathbf{X}_{mha}$ by:

$$\mathbf{X}_{ffn} = \text{LN}(\text{FC}(\text{GeLU}(\text{Conv}(\text{FC}(\mathbf{X}_{mha}))) + \mathbf{X}_{mha}), \quad (3)$$

where $\text{Conv}(\cdot)$ denotes the convolutional operations. Notably, SegFormer does not employ positional encodings inside the image patches but uses a convolutional layer $\text{Conv}(\cdot)$ to leak location information [16].

Finally, the MLP decoder transforms and concatenates the stage-wise feature maps $\{\mathbf{X}_i\}_{i=1}^4$ together to make the pixel-wise predictions. More details can be found in [6].

2.2. Motivation

The hierarchical visual representations in SegFormer also lead to the increasing width in the MiT encoder, which brings more computational burden on low-power devices.

However, we find that each neuron is not always informative across different input instances. Specifically, we take a trained SegFormer with MiT-B0 encoder to conduct inference over 2,000 instances of the ADE20K benchmark and observe neuron magnitudes across all testing instances. We show the box plot of neuron magnitudes of a certain MiT layer on the right side of Figure 1 The box plot contains the median, quartiles, and min/max values, which can help categorize these neurons into three types: *Type-I:* large median with a small range, meaning that the neuron is generally informative across different instances; *Type-II:* large median with a large range, meaning that the information of neuron highly depends on the input; *Type-III:* small median with a small range, meaning that these neurons are mostly non-informative regardless of different input.

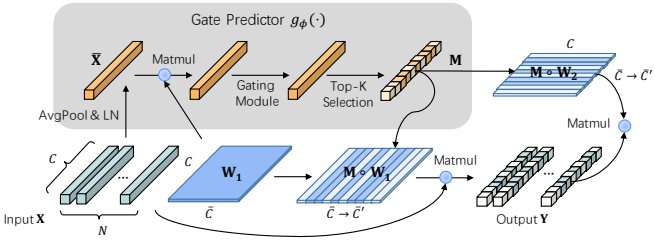

Fig. 2: Overview of our dynamic gated linear layer applied to two consecutive $\text{FC}(\cdot)$ layers parameterized by $\mathbf{W}_1$ and $\mathbf{W}_2$.

Given the above observation, it would be promising to reduce the computation inside the MiT encoder by identifying and pruning Type-II and Type-III neurons based on the input.

3. METHODOLOGY

3.1. Dynamic Gated Linear Layer

In order to identify and prune the un-informative neurons based on the input, we propose *Dynamic Gated Linear* (DGL) layer, a plug-and-play module to substitute the original linear layer in SegFormer. The DGL structure is shown in Figure 2.

The workflow of the DGL layer is as follows. Given input $\mathbf{X} \in \mathbb{R}^{N\times C}$, the DGL layer computes the gate $\mathbf{M} = g_\phi(\mathbf{X}) \in \mathbb{R}^{N\times \bar{C}}$ via a gate predictor $g_\phi(\cdot)$ parameterized by ϕ, where is $\bar{C}$ the output dimension. The gate $\mathbf{M}$ can be then applied to mask both the output dimensions of the current linear layer parameter $\mathbf{W}_1 \in \mathbb{R}^{C\times\bar{C}}$, as well as the input dimensions of the next layer parameter $\mathbf{W}_2 \in \mathbb{R}^{\bar{C}\times C}$. Therefore, the computation can be reduced for two consecutive layers as:

$$\mathbf{Y} = \mathbf{X}(\mathbf{W}_1 \circ \mathbf{M}) \in \mathbb{R}^{N\times\bar{C}}, \ \mathbf{Z} = \mathbf{Y}(\mathbf{W}_2 \circ \mathbf{M}^\top) \in \mathbb{R}^{N\times C},$$

where $\circ$ denotes the element-wise product.

The design of gate predictor $g_\phi(\cdot)$ is the key to achieve dynamic pruning. As the input the gate predictor, we first summarize the sequence by average pooling over the N image

patches of input $\mathbf{X}$, followed by layer-normalization to scale it back to normal ranges, i.e., $\hat{\mathbf{X}} = \text{LN}(\text{AvgPool}(\mathbf{X})) \in \mathbb{R}^{C}$. Intuitively, aside from the input $\mathbf{X}$, the parameter $\mathbf{W}$ should be also incorporated to determine which neuron to prune in the output. We thus feed both $\hat{\mathbf{X}}$ and $\mathbf{W}_1$ into a two-layer MLP$(\cdot)$ activated by ReLU$(\cdot)$ to obtain the mask $\mathbf{M}$ as

$$\mathbf{M} = \text{Top-r}(\mathbf{G}), \quad \mathbf{G} = \text{MLP}(\hat{\mathbf{X}}\mathbf{W}_1) \in \mathbb{R}^{\bar{C}},$$

where Top-r$(\cdot)$ keeps the top $r\%$ percentage largest elements in MLP output logits $\mathbf{G}$, and zero out the rest ones. In order to achieve a smooth transition to sparse parameters, we adopt an annealing strategy by gradually increasing the sparsity ratio at the t-th step as $r_t = r\min(1, \frac{t}{T})$, where T is total annealing steps. Note that the Top-r$(\cdot)$ operation inevitably introduces information loss. To remedy this, we also encourage sparsity over the $\mathbf{G}$, which cab be achieved by ℓ_1 norm penalty over $\mathbf{G}$ as the following:

$$\mathcal{L}_m = \lambda_m \sum \|\mathbf{G}\|_1. \tag{4}$$

We deploy the DGL layer to prune $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$, as well as the intermediate layer of the Mix-FFN of the SegFormer, as a majority of computation lies in the MiT Encoder. For the MLP decoder, we replace the concatenation by addition over $\{\mathbf{X}_i\}_{i=1}^{4}$, so that the computation can be further reduced.

Remark The gate predictor $g_\phi(\cdot)$ has only $O(C\bar{C})$ computational complexity, which is negligible compared with the $O(NC\bar{C})$ complexity in the linear layer. However, all model parameters together with gate predictors should be saved, as every single parameter can be potentially activated depending on the input. Finally, dynamic pruning has also been previously explored in [17] for CNNs, where they only pass input $\mathbf{X}$ to the gate predictor, missing the information inside the parameter $\mathbf{W}_1$ for the gate prediction.

3.2. Training with Knowledge Distillation

The dynamically pruned SegFormer inevitably loses the knowledge from the original model. To bridge the gap incurred by dynamic pruning, we adopt two-stage knowledge distillation [15, 18], which demonstrates superior performance on transformer-based models. In the first stage, we minimize the mean square error (MSE) between the student output $\tilde{\mathbf{X}}_i$ and the teacher output $\mathbf{X}_i$ at each SegFormer block, i.e.,

$$\ell_1 = \sum_{i=1}^{4} \text{MSE}(\tilde{\mathbf{X}}_i, \mathbf{X}_i). \tag{5}$$

Afterwards, with the student logits $\tilde{\mathbf{Y}}$, we minimize the conventional cross entropy loss (CE) with ground truth data $\mathbf{Y}^\star$ and soft-cross entropy (SCE) with teacher logits $\mathbf{Y}$ as:

$$\ell_2 = \text{CE}(\tilde{\mathbf{Y}}, \mathbf{Y}^\star) + \lambda_s \text{SCE}(\tilde{\mathbf{Y}}, \mathbf{Y}). \tag{6}$$

Note that for both two stages, we incorporate the sparsity regularization in Equation (4).

4. EXPERIMENTS

4.1. Experimental Setup

We empirically verify the proposed approach on ADE20K [19] and Cityscapes [20], two benchmark dataset for semantic segmentation. We follow the standard data pre-processing in [6], i.e., the images of ADE20K and Cityscapes are cropped to 512×512 and 1024×1024 respectively. We take the batch size as 16 on ADE20K, and 8 on Cityscapes, both of which are trained over 8 NVIDIA V100 GPUs. As we compare both the performance and efficiency of the model, we report the mean Intersection over Union (mIoU) together with computational FLOPs (G).

For the main experiments in Section 4.2, we present results for both real-time settings and non real-time settings. We adopt MiT-B0 as the encoder backbone for the real-time setting, and MiT-B2 for the non real-time setting. The dynamic pruning is based on the released model checkpoints [1]. During the training, we take the sparsity regularizer $\lambda_m = 0.005$, and soft cross-entropy regularizer $\lambda_s = 0.5$ throughout the experiments. The training iterations are set to 160K, where the first 50% steps are used for sparsity annealing. We keep the rest configurations consistent with ones used in SegFormer, e.g., with learning rate of 0.00006 under the "poly" LR schedule, and without auxiliary losses and class balance losses. We name the dynamically pruned SegFormer as DynaSegFormer.

4.2. Main Results

We present the main results in Table 1, where 30% and 50% neurons are dynamically pruned in the MiT encoder. It can be found that for both the real-time and non real-time settings on the two benchmarks, our DynaSegFormer can easily outperform previous CNN-based models with significantly fewer computational FLOPs. For instance, our DynaSegFormer outperforms DeepLabV3+ by 2.9% mIoU with only 5% of its FLOPs. While the DGL layer increases the size of SegFormer by around 12%, it allows to identify and prune the instance-wise redundant neurons given different input. For example, we can achieve 36.9% mIoU on ADE20K for the real-time setting, which is only 0.5% inferior to SegFormer using only 40% of the original computational FLOPs.

4.3. Discussions

For ablation studies, we adopt a MiT-B0 as the encoder under 50% sparsity for pruning. The model is trained with 40K iterations for fast verification, and results are present in Table 2. We analyze the following aspects: ***1) Knowledge Distillation:*** Comparing with training without distillation (row #4), two-stage distillation (row #1) can improve the mIoU by around 1.2%. However, distillation with only stage-1 or stage-2 cannot bring such improvement (rows #2, #3). The finding is consistent with [15, 18] that the technique can better transfer the

[1] https://github.com/NVlabs/SegFormer.

Table 1: Main results on the validation set of ADE20K and CityScapes. For DynaSegFormer results, $+0.44$ and $+3.35$ are the increased parameter with the proposed DGL layers.

	Models	Encoder	Params (M) ↓	ADE20K		Cityscapes	
				FLOPs (G)↓	mIoU (%)↑	FLOPs (G)↓	mIoU (%)↑
Real-Time	FCN [7]	MobileNet-V2	9.8	39.6	19.7	317.1	61.5
	ICNet [11]	-	-	-	-	-	67.7
	PSPNet [10]	MobileNetV2	13.7	52.9	29.6	423.4	70.2
	DeepLabV3+ [9]	MobileNetV2	15.4	69.4	34.0	555.4	75.2
	SegFormer [6]	MiT-B0	3.8	8.4	37.4	125.5	76.2
	DynaSegFormer	MiT-B0	$3.8_{+0.44}$	3.3	36.9	62.7	75.1
		MiT-B0	$3.8_{+0.44}$	2.7	35.0	46.7	74.3
Non Real-Time	FCN [7]	ResNet-101	68.6	275.7	41.4	2203.3	76.6
	EncNet [21]	ResNet-101	55.1	218.8	44.7	1748.0	6.9
	PSPNet [10]	ResNet-101	68.1	256.4	44.4	2048.9	78.5
	CCNet [22]	ResNet-101	68.9	255.1	45.2	2224.8	80.2
	DeepLabV3+ [9]	ResNet-101	62.7	255.1	44.1	2032.3	80.9
	OCRNet [23]	HRNet-W48	70.5	164.8	45.6	1296.8	81.1
	SegFormer [6]	MiT-B2	27.5	62.4	46.5	717.1	81.0
	DynaSegFormer	MiT-B2	$27.5_{+3.35}$	18.5	45.4	286.5	80.3
		MiT-B2	$27.5_{+3.35}$	15.0	44.6	214.2	79.8

Table 2: Ablation studies for pruning types, two-stage knowledge distillation and sparsity annealing.

#	Pruning Type	KD Stage-1	KD. Stage-2	Annealing Sparsity	mIoU (%)
1	Dynamic	✓	✓	✓	32.1
2	Dynamic	✓		✓	30.6
3	Dynamic		✓	✓	30.7
4	Dynamic			✓	30.9
5	Dynamic	✓	✓		31.4
6	Dynamic				30.4
7	Static	✓	✓	✓	29.2
8	Static			✓	27.7

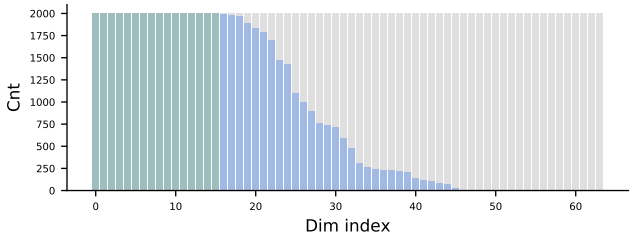

Fig. 3: Visualization of dynamic gating module on the key layer of stage-1.1 in MiT-B0. The green and blue bars refer to the Type-I and Type-II neurons. The Type-III neurons are absent as they are always not activated during the inference.

learned knowledge to the compressed model. ***2) Annealing Sparsity:*** It is found generally helpful to anneal the sparsity during the training. For instance, it can improve the mIoU by 0.7% for training with distillation (rows #1, #5), and 0.5% for training without distillation (row #4, #6). The observations match our intuition since a smooth transition of sparsity ratio can better calibrate the parameters throughout the training process. ***3) Dynamic v.s. Static Pruning***: Finally, we verify the advantages of dynamic pruning over static pruning. For static pruning, we follow the widely used magnitude-based pruning [24], where the sparse mask is invariant to different input. It can be found that even with both distillation and sparsity annealing, static pruning is still inferior to dynamic pruning by 2.9% (row #1, #7), while the gap is even enlarged to 3.2% when trained without distillation (row #4, #8).

In the next, we analyze how the learned DGL layer keeps the Type-I neurons, but identifies and prunes the Type-II and Type-III neurons, as mentioned in Section 2.2. We count the number of active gate logits generated by the DGL layer, i.e., $\sum_i \mathbb{I}(G_i > 0)$ during the inference, where $\mathbb{I}(\cdot)$ is the indicator function. From Figure 3, it can be found that there are around 15 neurons that are always activated in the $2,000$ testing instances on ADE20K, indicating that they are the Type-I neurons. The Type-II neurons can be selectively activated ranging from $10 \sim 1988$ counts, while the Type-III neurons are not visible since they are always inactive throughout all instances. Consequently, by identifying the types of neurons, the dynamic scheme selectively prunes neurons to better explain the data under limited computational constraints.

5. CONCLUSION

In this paper, we propose the dynamic gated linear layer to prune uninformative neurons in SegFormer based on the input instance. The dynamic pruning approach can be also combined with two-stage knowledge distillation to further improve the performance. Empirical results on benchmark datasets demonstrate the effectiveness of our approach.

6. REFERENCES

[1] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2021.

[2] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *ICML*, 2021, pp. 10347–10357.

[3] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," Preprint arXiv:2103.14030, 2021.

[4] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z. Jiang, F. Tay, J. Feng, and S. Yan, "Tokens-to-token vit: Training vision transformers from scratch on imagenet," Preprint arXiv:2101.11986, 2021.

[5] S. Zheng, J. Lu, H. Zhao, X. Zhu, Z. Luo, Y. Wang, Y. Fu, J. Feng, T. Xiang, P. Torr, et al., "Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers," in *CVPR*, 2021, pp. 6881–6890.

[6] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. Alvarez, and P. Luo, "Segformer: Simple and efficient design for semantic segmentation with transformers," in *NeurIPS*, 2021.

[7] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, 2015, pp. 3431–3440.

[8] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *TPAMI*, vol. 40, no. 4, pp. 834–848, 2017.

[9] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *ECCV*, 2018, pp. 801–818.

[10] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *CVPR*, 2017, pp. 2881–2890.

[11] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "Icnet for real-time semantic segmentation on high-resolution images," in *ECCV*, 2018, pp. 405–420.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *ECCV*, 2016, pp. 630–645.

[14] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," Preprint arXiv:1801.04381, 2018.

[15] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "Tinybert: Distilling bert for natural language understanding," in *Findings of EMNLP*, 2020.

[16] M. Islam, S. Jia, and N. Bruce, "How much position information do convolutional neural networks encode?," Preprint arXiv:2001.08248, 2020.

[17] X. Gao, Y. Zhao, L. Dudziak, R. Mullins, and C. Xu, "Dynamic channel pruning: Feature boosting and suppression," in *ICLR*, 2019.

[18] H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King, "Binarybert: Pushing the limit of bert quantization," in *ACL*, 2021.

[19] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *CVPR*, 2017, pp. 633–641.

[20] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *CVPR*, 2016, pp. 3213–3223.

[21] H. Zhang, K. Dana, Jianping Shi, Zhongyue Zhang, Xiaogang Wang, Ambrish Tyagi, and Amit Agrawal, "Context encoding for semantic segmentation," in *CVPR*, 2018, pp. 7151–7160.

[22] Z. Huang, X. Wang, L. Huang, C. Huang, Y. Wei, and W. Liu, "Ccnet: Criss-cross attention for semantic segmentation," in *ICCV*, 2019, pp. 603–612.

[23] Y. Yuan, X. Chen, and J. Wang, "Object-contextual representations for semantic segmentation," in *ECCV*. Springer, 2020, pp. 173–190.

[24] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *ICCV*, 2017, pp. 2736–2744.