Once For All Skip: Efficient Adaptive Deep Neural Networks

Yu Yang, Di Liu*, Hui Fang, Yi-Xiong Huang, Ying Sun and Zhi-Yuan Zhang
School of Software, Yunnan University
dliu@ynu.edu.cn, *Corresponding Author

***Abstract*—In this paper, we propose a new module, namely *once for all skip* (OFAS), for adaptive deep neural networks to efficiently control the block skip within a DNN model. The novelty of OFAS is that it only needs to compute once for all skippable blocks to determine their execution states. Moreover, since adaptive DNN models with OFAS cannot achieve the best accuracy and efficiency in end-to-end training, we propose a reinforcement learning-based training method to enhance the training procedure. The experimental results with different models and datasets demonstrate the effectiveness and efficiency in comparison to the state of the arts. The code is available at https://github.com/ieslab-ynu/OFAS.**

I. Introduction

Deep Neural Networks (DNNs) are the *de facto* core of modern artificial intelligence applications. To pursue higher accuracy, DNN models are becoming more complicated with more blocks and more channels. At the same time, to facilitate a responsive processing and protect data privacy, an increasing number of DNN models are deployed on edge/mobile devices to have in-situ processing [1].

Nevertheless, edge/mobile devices are subject to limited computation capability and memory space. Thus, many efforts have been made towards improving the efficiency of DNN models. For instance, efficient DNN architectures, e.g., MobileNet [2] and ShuffleNet [3], are proposed. Model pruning removes the redundant weights/channels of the complicated DNN models. Model quantization employs low-width bit representations to store the model's weights and do arithmetic operations, where 32-bit float-point (FP) numbers are quantized into binary, teneary or 4 bit integers [4]. Hardware-efficient neural architecture search (HwNAS) automates the tedious model design procedure and takes into account the target hardware to design efficient DNN models [5].

All the above-mentioned methods feature a static solution, i.e., the models are fixed and the run-time computations cannot be changed during its execution. Some works observe that for some simple images, the DNN models can skip some blocks within a model without degrading the prediction accuracy. Therefore, some works, e.g., [6], [7], propose to adaptively skip some blocks according to the input images so that the run-time computational complexity can be reduced. However, these methods use a complex auxiliary network, like RNN, reinforcement learning agent, to determine which blocks to be skipped. Although these methods can reduce the computation in comparison to the original model, the on-device inference latency does not decrease accordingly.

To address the issue of the existing adaptive DNNs, in this paper, we propose a new and light-weight control module to skip the execution blocks within a DNN model, namely *once-for-all-skip* (OFAS). Different from the existing methods, like SkipNet [6], our control module strives to determine the execution status of all blocks by only *one computation*, and thus our method not only reduces the computational complexity but also the inference latency. The OFAS module can be learned with the backbone network in an end-to-end fashion. However, training adaptive models are known to be difficult [8], and we find it is difficult to obtain the best adaptive models with end-to-end training. Thus, we propose a new training method to enhance the OFAS module training for better performance. The detailed contributions of our paper are as follows:

1) We propose the OFAS module to adaptively, effectively and efficiently skip the blocks of DNN models according to input images. The OFAS module can be trained with the backbone model in an end-to-end fashion;
2) We propose a reinforcement learning (RL) [9] based training method to enhance the training of OFAS module, such that the adaptive models equipped with OFAS can achieve better accuracy and efficiency;
3) We evaluate the OFAS method in comparison to the existing methods using different models and datasets. And the new OFAS method can reduce the latency by up 34% with small accuracy drop.

II. Related Work

Efforts are made towards efficient DNN models, like light-weight model design, by either hand-craft [2], [3] or neural architecture search (NAS) [10], [11] and model pruning [12], quantization, and knowledge distillation. Recently, some works observe that DNN models can skip some channels or blocks for 'simple' images without hurting their accuracy. Thus, some works strive to exploit this feature to have adaptive and efficient inference. Lin *et al.* [13] proposed a run-time and dynamic model pruning methods, where RL is used to select the skipped channels. However, the channel pruning does not always translate to the latency reduction, and RL causes some extra overhead. Therefore, the latency benefit from the run-time pruning is limited. Besides channels, model depth, i.e., the number of layers/blocks, can be changed for the adaptive inference. Two representative methods can achieve the depth adaptivity, i.e., early exit (EE) [7] and layer/block skipping [6].

978-3-9819263-6-1/DATE22/©2022 EDAA

Model	Latency(ms)	Accuracy(%)	Dataset
ResNet38 (Original)	34.3	92.77	cifar10
ResNet38+Gate	104.1	92.43	cifar10
ResNet38+Gate+RL	103.4	91.81	cifar10
ResNet38+RNN	105.6	92.11	cifar10
ResNet38+RNN+RL	98.3	91.93	cifar10

TABLE I: The results of SkipNet on Nvidia Jetson Nano, where each skippable block is controlled via the intermediate results from the previous block, thereby leading to higher latency.

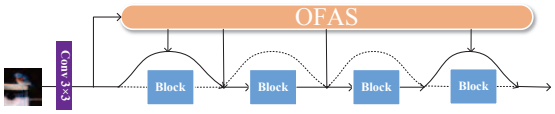


Fig. 1: The OFAS module takes as input the intermediate activation of the first convolutional layers and determines all skipping decisions.

OFAS-1	Size	OFAS-2	size
AvgPool	C_1	AvgPool	C_1
FC	$(C_1, C_1 \times 2)$	FC	(C_1, C_1)
FC	$(C_1 \times 2, n)$	FC	$(C_1, C_1 \times 3)$
Sigmoid	n	FC	$(C_1 \times 3, n)$
		Sigmoid	n

TABLE II: The configurations of two OFAS modules. n indicates the number of skippable blocks.

EE can predict the result using only the early blocks, and then the later blocks are skipped to reduce the latency. The flaw of EE is that it cannot exploit the fine-grained features provided by the later blocks, thus degrading the accuracy. SkipNet [6] is the representative of skipping. Skipping selectively executes some blocks and thus can exploit all fine-grained features from later blocks. However, the issue of SkipNet is the control overhead. Detailed analysis and results are provided in Section III. Motivated by this, we propose OFAS to mitigate the overhead issue.

III. Motivational Example

In this section, we show the flaws of existing adaptive methods, i.e., the skipped blocks do not translate to the latency reduction. We use the representative SkipNet [6] as an example. In SkipNet, the skip decision of the current block relies on the output of the precedent block. However, for the majority of hardware, DNNs are executed block by block, since there are strong correlation between two sequential blocks. As a result, adding many extra units to skip blocks will unnecessarily increase the latency. Table I shows the numerical results of ResNet38 [14] with different skip modules evaluated on Nvidia Jetson Nano. We can see that the models with different modules proposed in [6] (denoted as gate, gate+RL, RNN, RNN+RL) significantly increase the inference time by around 3x compared to the original model without the skip module while sacrificing the accuracy by 1%. This means although such skipping reduces the computational complexity (fewer operations are needed), it does not really translate to latency reduction which is more important for many latency-sensitive applications. This observation motivates us to seek for an efficient control module to mitigate the control overhead.

IV. The OFAS Module

To alleviate the decision overhead and convert the skipped blocks to the real latency reduction, we need to have a light-weight control unit. To this end, we propose the *once-for-all-skip* (OFAS) module which makes all skip decisions based on *one computation*. In this section, we present the details of the OFAS module, including its design philosophy and the new training method for the OFAS module.

A. OFAS Structure

The idea behind OFAS is to use one computation to determine the skip decisions for all blocks within the DNN model, thereby reducing the control overhead and achieving the latency reduction. Fig. 1 visualize the concept of OFAS, where OFAS takes as input activations of the first convolutional layer. We design two types of OFAS, OFAS-I with 2 FC layers for the shallow model and OFAS-II with 3 FC layers for the deep model (more than 50 layers like ResNet110). The output of the last FC layer is fed into a Sigmoid layer. Finally, we use the output from Sigmoid layer and a threshold T to determine the skip decisions. Table II shows the structures and hyper-parameters of the two OFAS modules, where n denotes the number of skippable blocks in the backbone model. Fig. 2 visualizes the structure of OFAS-I module.

We use a binary gate G_i to control the block skip, where 0 indicates to skip block i and 1 means to execute it. Since the output of Sigmoid is in range of $(0, 1)$, a threshold $T \in [0, 1)$ is given to evaluate the binary value. Formally, the skip decision for a gate is determined as:

$$G_i(x) = \begin{cases} 1 & \text{if } S_L(i) \geq T \\ 0 & \text{Others} \end{cases} \tag{1}$$

where

$$\text{OFAS-I}: \quad S_L = Sigmoid(\Pi_{j=1}^{2} \mathcal{F}_j(AvgPool(x))) \tag{2}$$

$$\text{OFAS-II}: \quad S_L = Sigmoid(\Pi_{j=1}^{3} \mathcal{F}_j(AvgPool(x))) \tag{3}$$

where S_L is a vector of n. $S_L(i)$ indicates the i^{th} element in S_L, i.e., the output from Sigmoid layer for the i^{th} skippable block. $\mathcal{F}_j$ denotes the j^{th} FC layer.

With the gate for the skippable block, the intermediate results of i^{th} block can be written as follows:

$$x_{i+1} = G_i(x)\mathcal{B}(x_i) + (1 - G_i(x))x_i \tag{4}$$

Remark: Threshold $T \in [0, 1)$ is a key element for OFAS and can be considered as a knob to tune the model between accuracy and efficiency. The bigger T is, the more blocks are skipped. We can train the OFAS module along with the backbone network in an end-to-end fashion. However, we find that training in the end-to-end fashion tends to learn similar decisions for different classes, thus prohibiting the module from exploring better skip decisions to improve the model's performance.

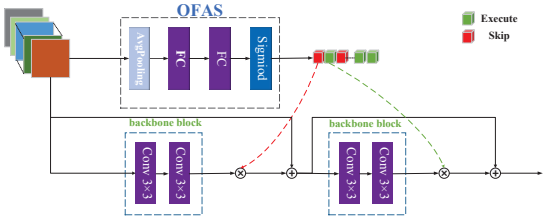


Fig. 2: The visualized structure of OFAS-I module.

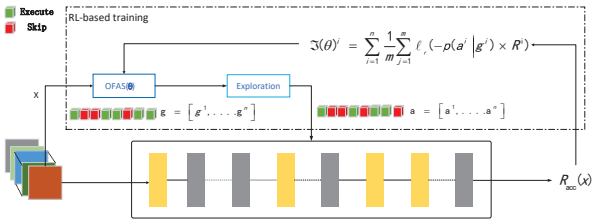


Fig. 3: The overview of RL-based training.

B. RL-Based Training

We propose an RL-based training method to complement the end-to-end training for our OFAS module. The RL-based training method is used after the end-to-end training. The motivation behind selecting RL is that RL is able to learn an optimal decision policy via continuous interactions and we find that our skipping decision improvement can be well formulated as an RL learning procedure. *Note that RL is only used for the training procedure, and it does not cause any extra inference overhead.*

We employ a policy gradient RL method [9], which, given a state, directly outputs the action and optimizes the policy. To train the model with RL, we directly take the OFAS module as the agent. Let π denote the decision policy learned and the policy can be written as $\pi\ (x;\theta)\rightarrow \boldsymbol{a}(x)$, where $\boldsymbol{a}(x)$ denotes the action under input x. θ is the learned parameters of π, i.e., the weights of the OFAS module. Thus, updating θ actually improves the decision policy of the OFAS module. Fig. 3 shows the overview of the RL-based training. A successful RL relies on 3 key ingredients: *action*, *state*, and *reward*.

1) State: State is the current representation of the environment, provided for the agent to make the optimal decision. In our case, the state is the intermediate results of the first convolutional layer, i.e., the input x for the OFAS module.

2) Action: Action represents a decision the agent makes upon the current state. Let $\boldsymbol{a}(x)$ represent a specific action with input x. In the context of OFAS, one action $\boldsymbol{a}(x)$ contains the skip states for all skippable blocks, i.e., $a(x)\in\{a^1(x), a^2(x), \cdots, a^n(x)\}$, where $a^i(x)$ represents the skip state of block i and n denotes the number of skippable blocks. For each skippable block, instead of directly taking the skip state $\boldsymbol{g}(x)$ from the OFAS module, we add one step to explore the different combinations for input x, such that the module is able to find a better skip combination.

$$a^i(x) = \begin{cases} g^i(x) & \text{with probability } 0.9 \\ 1 - g^i(x) & \text{with probability } 0.1 \end{cases} \tag{5}$$

where $g^i(x)$ is the skip state of gate i under input x derived by the OFAS module (see Fig. 3). Eq. (5) indicates each gate has a small probability of 0.1 to change the gate state and this allows the module to explore different possibilities, thus preventing the module from converging to a suboptimal result.

3) Reward: Reward is the feedback value the agent receives after it makes an action. RL agent deploys the returned reward to update θ to optimize the decision policy. In our setting, we devise a new reward function for our RL-based training. For each gate i, the reward is computed as follows:

$$R^i(x) = \beta\left((1-a^i) - \gamma_1 R_{acc}(x) + \gamma_2 R^{i+1}(x)\right) \tag{6}$$

where β, similar to learning rate, is a small value used to control the update step and we empirically set β to $0.1/n$. $\gamma_1 = 0.3$ and $\gamma_2 = 0.9$ are two discount factors. R_{acc} represents the reward for accuracy, which is defined as:

$$R_{acc}(x) = \mathcal{I}(\hat{y}(x), y) \tag{7}$$

$$\hat{y}(x) = -\log(\mathcal{N}(x, \boldsymbol{a}(x)) \tag{8}$$

where $\mathcal{I}(\cdot)$ is the summation of element-wise multiplication. Label y is one-hot vector like $[0,0,0,1]$, and $R_{acc}(x)$ is a scalar. For example, assume $\hat{y}(x) = [2.302, 1.609, 1.203, 0.916]$ and $y = [0,0,0,1]$, then $R_{acc}(x)$ is 0.916. $\mathcal{N}(x, \boldsymbol{a}(x))$ represents the predicted probability distribution under action $\boldsymbol{a}(x)$. $\hat{y}(x)$ indicates the negative logits of the predicted probability distribution. The higher the predicted accuracy is, the smaller the R_{acc} is. Since the decision of the current gate may also affect the decisions of the subsequent blocks, we use $R^{i+1}(x)$ to account for the effect on the subsequent blocks.

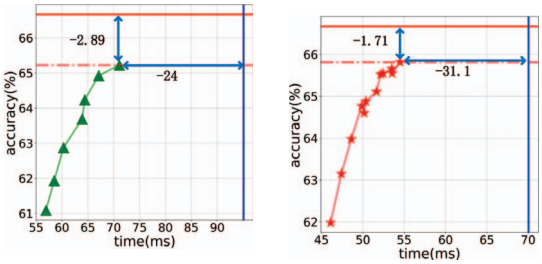
With all elements, we define the following loss function to update policy π.

$$\begin{aligned} \min \mathcal{J}(\theta) &= \min E_L \overline{E}_X L_\theta(a, x) \\ &= \min \sum_{i=1}^{n} \frac{1}{m} \sum_{j=1}^{m} \ell_r(-p^i(j) \times R^i(j)) \end{aligned} \tag{9}$$

where ℓ_r represents the learning rate and

$$p^i(j) = \begin{cases} 0.9 & \text{if } \ a^i(j) = g^i(j) \\ 0.1 & \text{others} \end{cases} \tag{10}$$

where n represents the number of skippable blocks, and m represents the batch-size. $p^i(j)$ is a parameter to adjust the loss value and stabilize the training process. For the gate which does not change its state from $g^i(j)$ (i.e., the OFAS output), we can have a large loss value. However, if the gate's state is changed, using the original value as the loss function will drastically change the weights, thereby making the training difficult to converge. Hence, we use a small constant to limit this weight update.



(a) cifar100-ResNet-110 (wo RL) (b) cifar100-ResNet-110 w RL

Fig. 4: Experimental results for ResNet-110 w/wo RL-based training. The blue line indicates the latency of the original model and the red line indicates the accuracy of the original model without skipping blocks.

Model	Latency(ms)	Accuracy(%)	Dataset
ResNet38 (Original)	34.3	**92.77**	cifar10
ResNet38+SkipNet	104.1	**92.43**	cifar10
ResNet38+Early Exit	31.5	89.81	cifar10
ResNet38+OFAS+RL	**29.5**	91.8	cifar10
ResNet38 (Original)	33.8	69.55	cifar100
ResNet38+SkipNet	103.7	69.31	cifar100
ResNet38+Early-Exit	38.3	67.07	cifar100
ResNet38++OFAS+RL	**33.3**	**69.60**	cifar100
ResNet110 (Original)	90.1	71.34	cifar100
ResNet110+SkipNet	278	**70.83**	cifar100
ResNet110+Early-Exit	72.5	67.17	cifar100
ResNet110+OFAS+RL	**59.0**	69.63	cifar100

TABLE III: The summary of all experimental results

V. Experimental Evaluation

In this section, we evaluate the proposed OFAS modules in terms of accuracy and efficiency, i.e., latency. We use two models as the backbone network, ResNet38 and ResNet110 from the widely-used ResNet family [14]. Cifar10/100 datasets are used for the model training and validation. ResNet38 is with OFAS-I and ResNet110 is with OFAS-II. Since the increasing number of DNN models is implemented on edge/embedded systems, the latency is measured on Nvidia Jetson Nano, a representative edge platform with limited resource. For experimental results, we report the average latency for all validation images. We compare OFAS with SkipNet [6] and Early-Exit [7]. T in Eq. (1) is set to 0.5 for all experiments.

A. Comparison to the state of the arts

In this section, we compare OFAS with SkipNet and Early-Exit in terms of latency and accuracy. All the experimental results are summarized in Table III.

- SkipNet has relatively higher accuracy comparing to other methods, because it has a more fine-grained control for each block. However, such fine-grained control comes at the cost of significantly increased latency.
- Early-Exit is able to reduce the inference latency in two cases. However, since it cannot utilize the fine-grained features provided by the later blocks, the accuracy drops more than other two methods. For ResNet110 with Cifar100, the accuracy drops by 4.17%.
- OFAS strikes a good balance between accuracy and efficiency. In the worst case of ResNet110 with Cifar100, OFAS drops the accuracy by 1.71% but reduces the latency by 34% from 90.1ms to 59ms.

B. The Effectiveness of RL-based training

This section shows the evaluation of the RL-based training method proposed in Section IV-B, where we train ResNet110 with Cifar100 and the model is trained 500 epochs in the end-to-end training. Other settings remain the same. In this experiment, we adjust threshold T to trade off accuracy for latency. This result demonstrates the relationship between accuracy and latency, and it can be used for temporal adaptivity. The experimental results are shown in Fig. 4. The RL-based training method can benefit the OFAS module in both accuracy and efficiency. The model trained with RL method has smaller latency with higher accuracy.

VI. Conclusion

The low efficiency of the existing adaptive DNN models motivates this work. A new control module, namely OFAS, is proposed. The core feature of OFAS is to only compute once for all skip decisions, so that the real latency reduction can be achieved. Training adaptive DNN models is difficult and sometimes ineffective, so we propose an RL-based training method for OFAS to improve its accuracy and efficiency. The experimental results show the superiority of the newly proposed OFAS in terms of accuracy and latency.

Acknowledgement

This work was supported by NSFC under Grant 61902341, funds from Yunnan province under Grant 202101AT070182, 202101AS07007 and 2020SE403.

References

[1] D. Liu *et al.*, "Bringing ai to edge: From deep learning's perspective," *arXiv preprint arXiv:2011.14808*, 2020.
[2] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of CVPR*, 2018, pp. 4510–4520.
[3] N. Ma *et al.*, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of ECCV*, 2018, pp. 116–131.
[4] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," *Proceedings of ICLR*, 2018.
[5] X. Luo *et al.*, "Hsconas: Hardware-software co-design of efficient dnns via neural architecture search," in *Proceedings of DATE)*, 2021, pp. 418–421.
[6] X. Wang *et al.*, "Skipnet: Learning dynamic routing in convolutional networks," in *Proceedings of ECCV*, 2018, pp. 409–424.
[7] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *Proceedings of ICPR*, 2016, pp. 2464–2469.
[8] H. Li *et al.*, "Improved techniques for training adaptive deep networks," in *Proceedings of ICCV*, 2019, pp. 1891–1900.
[9] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[10] X. Luo *et al.*, "Edgenas: Discovering efficient neural architectures for edge systems," in *Proceedings of ICCD)*. IEEE, 2020, pp. 288–295.
[11] M. Tan *et al.*, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of CVPR*, 2019, pp. 2820–2828.
[12] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *ICLR*, 2015.
[13] J. Lin *et al.*, "Runtime neural pruning," in *Proceedings of NIPS*, 2017, pp. 2178–2188.
[14] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of CVPR*, 2016, pp. 770–778.