S2DNAS: Transforming Static CNN Model for Dynamic Inference via Neural Architecture Search

Zhihang Yuan[1,2*], Bingzhe Wu[1*], Guangyu Sun[1,2**], Zheng Liang[1],
Shiwan Zhao[3], and Weichen Bi[1]

[1] Peking University, Beijing, China
`yuanzhihang,wubingzhe,gsun,liangzheng,biweichen@pku.edu.cn`
[2] Advanced Institute of Information Technology, Peking University, Hangzhou, China
[3] IBM China Research Laboratory, Beijing, China
`zhaosw@cn.ibm.com`

Abstract. Recently, dynamic inference has emerged as a promising way to reduce the computational cost of deep convolutional neural networks (CNNs). In contrast to static methods (e.g., weight pruning), dynamic inference adaptively adjusts the inference process according to each input sample, which can considerably reduce the computational cost on "easy" samples while maintaining the overall model performance.

In this paper, we introduce a general framework, S2DNAS, which can transform various static CNN models to support dynamic inference via neural architecture search. To this end, based on a given CNN model, we first generate a CNN architecture space in which each architecture is a multi-stage CNN generated from the given model using some predefined transformations. Then, we propose a reinforcement learning based approach to automatically search for the optimal CNN architecture in the generated space. At last, with the searched multi-stage network, we can perform dynamic inference by adaptively choosing a stage to evaluate for each sample. Unlike previous works that introduce irregular computations or complex controllers in the inference or re-design a CNN model from scratch, our method can generalize to most of the popular CNN architectures and the searched dynamic network can be directly deployed using existing deep learning frameworks in various hardware devices.

Keywords: dynamic inference; neural architecture search; CNN

1 Introduction

In the past years, deep convolutional neural networks (CNNs) have gained great success in many computer vision tasks, such as image classification [20, 12, 17], object detection [34, 32, 29], and image segmentation [11, 4]. However, the remarkable performance of CNNs always comes with huge computational cost,

---

* Equal contribution
** Corresponding author

which impedes their deployment in resource constrained hardware devices. Thus, various methods have been proposed to improve computational efficiency of the CNN inference, including network pruning [21, 10, 22] and weight quantization [8, 44, 31]. Most of the previous methods are static approaches, which use fixed computation graphs for all test samples.

Recently, dynamic inference has emerged as a promising alternative to speed up the CNN inference by dynamically changing the computation graph according to each input sample [39, 2, 6, 5, 45, 16, 28, 7]. The basic idea is to allocate less computation for "easy" samples while more computation for "hard" ones. As a result, the dynamic inference can considerably save the computational cost of "easy" samples without sacrificing the overall model performance. Moreover, the dynamic inference can naturally exploit the trade-off between accuracy and computational cost to meet varying requirements (e.g., computational budget) in real-world scenarios.

To enable the dynamic inference of a CNN model, most previous works aim to develop dedicated strategies to dynamically skip some computation operations during the CNN inference according to different input samples. To achieve this goal, these works attempted to add extra controllers in-between the original model to select which computations are executed. For example, well-designed gate-functions were proposed as the controller to select a subset of channels or pixels for the subsequent computation of the convolution layer [7, 5, 14]. However, these methods lead to irregular computation at channel level or spatial level, which are not efficiently supported by existing software and hardware devices [42, 46, 15]. To address this issue, a more aggressive strategy that dynamically skips whole layers was proposed for efficient inference [45, 41, 40]. Unfortunately, this strategy can only be applied to the CNN model with residual connection [12]. Moreover, the controllers of some methods comes with a considerable complex structure, which cause the increase of the overall computational cost in the inference (see experimental results in Section 4).

To mitigate these problems, researchers propose early exiting the "easy" input samples at inference time [30, 39, 16, 1]. A typical solution is to add intermediate prediction layers at multiple layers of a normal CNN model, and then exit the inference when the confidence score of the intermediate classifier is higher than a given threshold. Figure 1a shows the paradigm of these early exiting methods [30, 1]. In this paradigm, prediction layers are directly added in-between the original network and the network is split into multiple stages along the layer depth. However, these solutions face the challenge that early classifiers are unable to leverage the semantic-level features produced by the deeper layers. It may cause a significant accuracy drop [16]. As illustrated in Figure 1a, three prediction layers are added to different depth of the network. Thus, the classifier[4] in the previous stage cannot make use of the semantic-level features produced by the classifier in the late stage.

Huang et al. [16] proposed a novel CNN model, called MSDNet, for solving this issue. The core design of MSDNet is a two-dimensional multi-scale archi-

---

[4] In this paper, the classifier refers to the whole sub-network in the current stage.
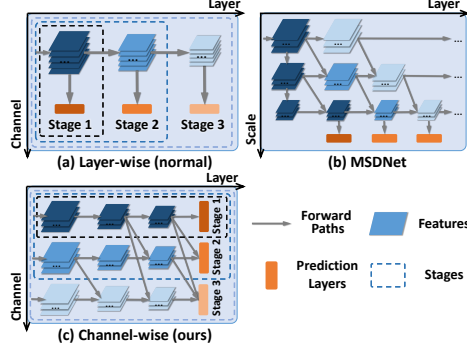
Fig. 1. Three paradigms of early exiting methods. (a) The layer-wise approach splits the network into multiple stages along the layer depth. (b) MSDNet devises a multi-stage CNN in which each stage maintains a feature pyramid. (c) Our proposed channel-wise approach splits the network along the channel width.

tecture that maintains the coarse and fine level features in every layer as shown in Figure 1b. Based on this design, MSDNet can leverage the semantic-level features in every prediction layer and achieve the best result. However, MSD-Net needs to design specialized network architecture, which cannot generalize to other CNN models and needs massive expertise in architecture design.

To solve the aforementioned issue without designing CNNs from scratch, we propose to transform a given CNN model into a *channel-wise* multi-stage network, which comes with the advantage that the classifier in the early stages can leverage the semantic-level features. Figure 1c intuitively demonstrates the idea behind our method. Different from the normal paradigm in Figure 1a, our method split the original network into multiple stages along the channel width. The prediction layers are added only to the last convolutional layer, thus all classifiers can leverage the semantic-level features. To reduce the computational cost of the classifiers in the early stages, we propose to cut down the number of channels of each layer in different stages (more details can be found in Section 3).

Based on the high-level idea introduced above, we present a general framework called S2DNAS. Given a specific CNN model, the framework can automatically generate the dynamic model following the paradigm showed in Figure 1c. S2DNAS consists of two components: S2D and NAS. First, the component S2D, which means "static to dynamic", is used to generate a CNN model space based on the given model. This space comprises of different multi-stage CNN networks generated from the given model based on the predefined transformations. Then, NAS is used to search for the optimal model in the generated space with the help of reinforcement learning. Specifically, we devise an RNN to decide the setting of each transformation for generating the model. To exploit trade-off between accuracy and computational cost, we design a reward function that can reflect both the classification accuracy and the computational cost inspired by the prior works [38, 13, 43]. We then use a policy-gradient based algorithm [36] to train the

RNN. The RNN will generate better CNN models with reinforcement learning and we can further use the searched model for dynamic inference.

To verify the effectiveness of S2DNAS, we perform extensive experiments by applying our method to various CNN models. With a comparable model accuracy, our method can achieve further computation reduction in contrast to the previous works for dynamic inference.

2   Related Work

Static Method for Efficient CNN Inference. Numerous methods are proposed for improving the efficiency of CNN inference. Two representative research directions are network pruning [21, 10, 9, 22] and quantization [8, 44, 25, 31]. Specifically, network pruning aims to remove redundant weights in a well-trained CNN without sacrificing the model accuracy. In contrast, network quantization aims to reduce the bit-width of both activations and weights. Most works in the above two directions are static, which refers to using the same computation graph for all test samples. Next, we introduce an emerging direction of utilizing dynamic inference for improving the efficiency of CNN inference.

Dynamic Inference. Dynamic inference also refers to adaptive inference in previous works [40, 23]. Most previous works aim to develop dedicated strategies to dynamically skip some computation during inference. They attempted to add extra controllers to select which computations are executed [5, 3, 33, 24, 14, 7, 45, 40, 41]. Dong et al. [5] proposed to compute the spatial attention using extra convolutional layers then skipping the computation of inactive pixels. Gao et al. [7] proposed to compute the importance of each channel then skipping the computation of those unimportant channels. However, these methods lead to irregular computation at channel level or spatial level, which is not efficiently supported by existing deep learning frameworks and hardware devices. To address this issue, a more aggressive strategy that dynamically skips the whole layers or blocks is proposed [45, 40, 41]. For example, BlockDrop [45] introduced a policy network to decide which layers should be skipped. Unfortunately, this strategy can only be applied to the CNN model with residual connection. Moreover, these methods introduce extra controllers into the computational graph, the computational cost will remain the same or even increase in some cases. On the other hand, early exiting methods propose to divide a CNN model into multiple stages and exit the inference of "easy" samples in the early stages [30, 39, 16, 1]. The state-of-the-art is MSDNet [16] in which the authors manually design a novel multi-stage network architecture to serve the purpose of dynamic inference.

Neural Architecture Search. Recently, neural architecture search (NAS) has emerged as a promising direction to automatically design the network architecture to meet varying requirements of different tasks [47, 48, 13, 26, 43, 27]. There are two typical types of works in this research direction, RL-based searching algorithms [47] and differentiable searching algorithms [27]. In this paper, according
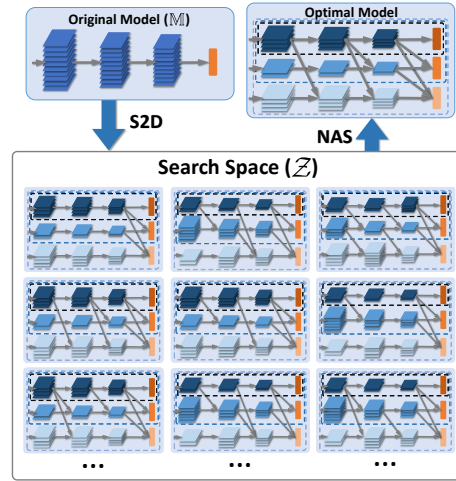
Fig. 2. Overview of S2DNAS. `S2D` first generates a search space from the original CNN model. Then, `NAS` searches for the optimal model in the generated space.

to the formulation of our specific problem, we choose the RL-based searching algorithm to search for the optimal model in a design space.

3 Our Approach

3.1 Overview of S2DNAS

The overview of S2DNAS is depicted in Figure 2. At a high level, S2DNAS can be divided into two components, namely, `S2D` and `NAS`. Here, `S2D` means "static-to-dynamic", which is used to generate a search space comprises of dynamic models based on a given static CNN model. Specifically, we define two transformations and then apply the transformations to the original model for generating different dynamic models in the search space. Each of these dynamic models is a multi-stage CNN that can be directly used for dynamic inference. All these generated models form the search space. Once the search space is generated, `NAS` searches for the optimal model in the space. In what follows, we will give the details of these two components.

3.2 The Details of `S2D`

Given a CNN model $\mathbb{M}$, the goal of `S2D` is to generate the search space $\mathcal{Z}$ which consists of different dynamic models transformed from $\mathbb{M}$. Each network in $\mathcal{Z}$ is a multi-stage CNN model in which each stage contains one classifier. These multi-stage CNNs can be generated from $\mathbb{M}$ using two transformations, namely, `split` and `concat`. First, we propose `split` to split the original model along
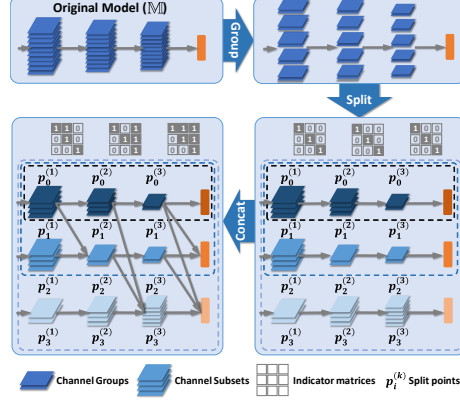
Fig. 3. Illustration of how `split` and `concat` are applied to a CNN model. Note that Group is an intermediate step of `spilt` for reducing the size of search space (see more details in the main text).

the channel width as Figure 3 shows. Specifically, we divide the input channels in each layer of the original model into different subsets. And each classifier can use features from different subsets for prediction. The prediction can be done by adding a prediction layer (shown as yellow squares in Figure 3). Moreover, to enhance the feature interactions between different stages for further performance boost, we propose `concat` to enforce the classifier in the current stage to reuse the features from previous stages. Next, we will present the details of these two transformations, `split` and `concat`. Before that, we first present some basic notations.

Notation. We start with the notation of a normal convolutional layer. Taking the k-th layer of a deep CNN as an example, the input of the k-th layer is denoted as $\mathbf{X}^{(k)} = \{X_1^{(k)}, \cdots, X_C^{(k)}\}$, where C is the number of input channels and $X_i^{(k)}$ is the i-th feature map. We denote the weights of layer k as $w^{(k)} = \{w_1^{(k)}, \cdots, w_O^{(k)}\}$, where O is the number of output channels and $w_i^{(k)} \in \mathbb{R}^{k_c \times k_c \times C}$ ($k_c \times k_c$ is the kernel size). In the following parts, we will present two transformations that can be applied to the original model. The goal of the transformations is to transform a static CNN model $\mathbb{M}$ to a multi-stage model, which can be represented as $a = \{f_1, \cdots, f_s\}$, where f_i is the classifier in the i-th stage. Next, we will introduce the details of the proposed two transformations.

Split. The `split` transformation is responsible for assigning different subsets of the input channels to the classifiers in different stages. We denote the number of stages as s. A direct way is splitting the input channels into s subsets and allocating the i-th subset to the classifier in the i-th stage. However, this splitting method results in a considerable large search space which poses the obstacle to the subsequent search process (i.e., `NAS`). In order to reduce the search space

generated by this transformation, we propose to first divide the input channels into groups and then assign these groups to different classifiers.

Specifically, we first evenly divide the input channels[5] into G groups thus each group consists of $m = \dfrac{C}{G}$ input channels. Taking the k-th layer as an example, the i-th group consists $\{X_{(i-1)m+1}^{(k)}, \cdots, X_{im}^{(k)}\}$. Once the grouping is finished, these groups are assigned to the classifiers in different stages. Precisely, we use the split points $(p_0^{(k)}, p_1^{(k)}, \cdots, p_{s-1}^{(k)}, p_s^{(k)})$ to split the groups, here $p_0^{(k)} = 0$ and $p_s^{(k)} = G$ are two peculiar points, which denote the start and end points. With the split points, we assign the groups between $p_{i-1}^{(k)}$ and $p_i^{(k)}$ to the classifier f_i in the i-th stage. Note that the connection (of the original model $\mathbb{M}$) between different classifiers are removed (see Figure 3).

Concat. The `concat` transformation is used for enhancing the interaction between different stages. The basic idea is to enable the classifiers in later stages to reuse the features from previous stages. Formally, we use indicator matrices $\{\mathbf{I}^{(k)}\}_{k=1}^{L}$ to indicate whether to enable the feature reuse at different positions. Here k denotes the k-th layer and L is the depth[6] of the CNN model. The element $m_{ij}^{(k)} \in \mathbf{I}^{(k)}$ indicates whether to reuse the features of the i-th stage in the j-th stage at the k-th layer, i.e., $m_{ij}^{(k)} = 1$ means that the classifier in the j-th stage will concat all the feature maps (of k-th layer) from the i-th stage. Note that we restrict the previous stages from concat the features of the later stages, i.e., $m_{ij}^{(k)} = 0, j < i, \forall k < L$. Moreover, we force the L-th layer (the prediction layer[7]) to concat the features from all the previous stages. We demonstrate a concrete example in Figure 3 to illustrate how to use the above two transformations to reshape a CNN model.

Architecture Search Space. Based on the above two transformations, we can generate the search space by transforming the original CNN model. Specifically, there are two adjustable settings for the two transformations, splitting points and indicator matrices. Adjusting the splitting points will change the way to assign the feature groups, which is used for the trade-off between accuracy and computational cost of different classifiers. For example, we can assign more features to the early stages for improving the model performance on "easy" samples. Adjusting the indicator matrices accompanies the change of the feature reuse strategy. To reduce the size of the search space, we restrict the feature layers with the same resolution to use the same split and concat settings in our experiments. Through changing these two settings, we can generate the search space $\mathcal{Z}$ which consists of different multi-stage models. In the following section, we will demonstrate how to search for the optimal model in the generated space.

---

[5] We do not split the first layer.

[6] Omit the batch normalization and pooling layers.

[7] Refer to the last layer of the classifier for prediction.

3.3   The Details of `NAS`

Once we obtain the search space $\mathcal{Z}$ from the above procedure of `S2D`, the goal of `NAS` is to find the optimal model a with high accuracy and low computational cost. Note that the model is jointly determined by the settings of the above two transformations, i.e., the split points and the indicator matrices. With a slight abuse of notation, we also refer the architecture a as these two settings and denote $\mathcal{Z}$ as the space which consists of these different settings. Thus the optimization goal reduces to search for the optimal settings of the proposed transformations which can maximize our predefined metric (see details in the following section).

However, searching the optimal setting is nontrivial due to the huge search space $\mathcal{Z}$. For example, in our experiment on MobileNetV2 [35], the size of the search space is around 10^{11}. Motivated by the recent progress in neural architecture search (NAS) [47, 48, 13, 38], we propose to use a policy gradient based reinforcement learning algorithm for searching. The goal of the algorithm is to optimize the policy π which further proceeds the optimal model. This process can be formulated into a nested optimization problem:

$$
\begin{aligned}
\arg\max_{\pi} \ & \mathbb{E}_{a \sim \pi} \left(R(a, W_a^*, \mathcal{D}_{val}) \right) \\
\text{s.t. } & W_a^* = \arg\min_{W_a} \mathcal{L}(a, W_a, \mathcal{D}_{train}) \ ,
\end{aligned}
\tag{1}
$$

where W_a is the corresponding weights of the model a and π is the policy which generates the settings of the transformations. $\mathcal{D}_{val}$ and $\mathcal{D}_{train}$ denote the validation and training datasets, respectively. And R is the reward function for evaluating the quality of the multi-stage model.

To solve the nested optimization problem in Equation 1, we need to solve two sub-problems, namely, optimizing π when W_a^* is given and optimizing W_a when the architecture a is given. We first present how to optimize the policy π when W_a^* is given.

Optimization of the Transformation Settings. Similar to previous works [47, 48], we use a customized recurrent neural network (RNN) to generate the distribution of different transformation settings for each layer of the CNN model. Then a policy gradient based algorithm [36] is used for optimizing the parameters of the RNN to maximize the expected reward, which is defined in Equation 2. Specifically, the reward in our paper is defined as a weighted product considering both the accuracy and the computational cost:

$$
R(a, W_a, \mathcal{D}) = \mathrm{COST}(a, W_a, \mathcal{D})^{\omega} \times \mathrm{ACC}(a, W_a, \mathcal{D}) \ ,
\tag{2}
$$

where $\mathrm{COST}(a, W_a, \mathcal{D})$ is the average computational cost over the samples of the dataset $\mathcal{D}$ using dynamic inference (i.e. $\frac{1}{|\mathcal{D}|} \sum_i \mathrm{COST}_i \times N_i$, where COST_i is the cost of f_i and N_i is the number of samples exiting at f_i). For a fair
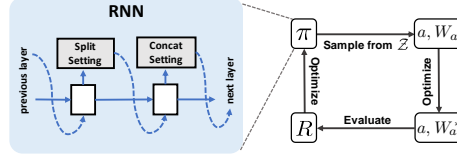
Fig. 4. The process of `NAS`. The RNN model is responsible for outputting the policy π, i.e., settings of `split` and `concat`, which further produces a model a. We can then optimize a for approximating the optimal parameter W_a^*. The computational cost and the accuracy of a is used from evaluating the generated policy π.

comparison with other works of dynamic inference, we use FLOPs [8] as the proxy of the computational cost. ω is a hyper-parameter which can be used for controlling the trade-off between model performance and the computational cost. The $\text{ACC}(a, W_a, \mathcal{D})$ is the accuracy of the multi-stage model on the dataset $\mathcal{D}$ (i.e. $\frac{1}{|\mathcal{D}|} \sum_i \text{ACC}_i \times N_i$, where ACC_i is accuracy of the N_i samples exiting at f_i). Next, we will introduce how to solve the inner optimization problem, i.e., optimizing W_a on the training dataset when the model a is given.

Optimization of the Multi-stage CNN. The inner optimization problem (i.e., solving for W_a^*) can be solved using the gradient descent algorithm. Specifically, we modify the normal classification loss function (i.e., cross-entropy function) for the case of training multi-stage models. Formally, the loss function is defined as:

$$\mathcal{L} = \sum_{(x,y) \in D_{train}} \sum_{i=1}^{s} \alpha_i \text{CE}(f_i(x, W_a), y) , \qquad (3)$$

Here, CE denotes the cross-entropy function, x is the input image and y is the class label. The optimization of the above equation can be regarded as jointly optimizing all the classifiers in different stages. The optimization can be implemented using stochastic gradient descent (SGD) and its variants. We use the optimized W_a for assessing the quality of the model a generated by the RNN, which can be further used for optimizing the RNN. In practice, to reduce the search time, following the previous work [48], we approximate W_a^* by updating it for only several training epochs, without solving the inner optimization problem completely by training the network until convergence.

Dynamic Inference of the Searched CNN. Once the optimal multi-stage model $a = \{f_1, \cdots, f_s\}$ is found, we can directly perform dynamic inference using it. Specifically, we set a predefined threshold for each stage. Formally, the threshold of the i-th stage is set to t_i. Then, we can use these thresholds to decide at which stage that the inference should stop. Specifically, given a input sample x, the inference stops at the i-th stage when the i-th classifier outputs a

---

[8] Here, we regard one multiply-accumulate (MAC) as one floating-point operation (FLOP).

top-1 confidence score $c_i \geq t_i$, here, $c_i = \max(f_i(x))$. Note that the threshold of the classifier should be set before we perform dynamic inference. We use the grid search to find the best threshold on training samples. At first, the correctness and confidences of the samples are obtained by inference them on all the s classifiers. Then we set different thresholds of classifiers from the grid of the thresholds and calculate the rewards on training data set using Equation 2. The thresholds with the highest reward are choose.

4   Experiments

To verify the effectiveness of S2DNAS, we compare it with different dynamic inference methods on different CNN models. Our experiments have covered a wide range of previous methods of dynamic inference [5, 45, 30, 39]. We also evaluate different aspects of S2DNAS, which are presented in the discussion part.

4.1   Experiment Settings

Model Setup. In our experiments, we conduct experiments on three CNN architectures: ResNet [12], VGG [37], and MobileNetV2 [35] [9]. Moreover, to compare with MSDNet, we use the DenseNet-like model and perform S2DNAS on the model.

Searching Details. The CIFAR [19] dataset contains 50k training images and 10k test images. We randomly choose 5k images from the training images as the validation dataset and leave the other 45k images as the training dataset. We use the same input preprocessing for both CIFAR-10 and CIFAR-100. To be specific, the training images are zero-padded with 4 pixels and then randomly cropped to 32x32 resolution. The randomly horizontal flip is used for data augmentation.

For the training of the RNN, the PPO algorithm [36] is used. And we use Adam [18] as the optimizer to perform the parameter update in RNN. For the training of the multi-stage model, we use SGD as the optimizer. The momentum is set to 0.9. The initial learning rate is set to 0.1 and the learning rate is divided by a factor of 10 at 50% and 75% of the total epochs.

For the hyper-parameters of S2DNAS, we set the group number $G = 8$ for every layer. And we set the number of stages $s = 3$. For comparing with MSDNet which contains 5 stages, we set the $s = 5$ for performing S2DNAS on DenseNet-like model. The ω in Equation 2 is set to -0.06 and all of the α_i in Equation 3 is set to 1 for all experiments.

4.2   Classification Results

In this part, we compare our method with other methods of dynamic inference. To give a comprehensive study of our method, we have covered a wide range of

---

[9] We use the batch normalization after each convolution layer in VGG and change the stride of the first convolution layer in MobileNetV2 from 2 to 1 for CIFAR.

methods, including LCCL [5], BlockDrop [45], Naive [30] and BranchyNet [39]. We conduct experiments on two widely-used image classification benchmarks, CIFAR-10 and CIFAR-100. To show the effectiveness of S2DNAS in reducing the computational cost of CNN models with different architectures, we apply S2DNAS to five typical CNNs with various depth, width, and sub-structures.

The overall results are shown in Table 1. Note that different thresholds (t_i defined in the previous section) lead to different trade-offs between model accuracy and the computational cost. In our experiments, we chose the threshold which leads to the highest reward on the validation dataset. We also provide further results of using different thresholds in the discussion subsection.

Table 1. Evaluations on CNN models with different architectures. The number in the Reduction column denotes the relative cost reduction compared with the original model. Some results are missing because there is no implementation of these CNN models in the reference papers.

Model	Method	CIFAR-10			CIFAR-100		
		FLOPs	Reduction	Accuracy	FLOPs	Reduction	Accuracy
ResNet-20	Original	41M	-	91.25%	41M	-	67.78%
	LCCL	30M	28%	90.95%	40M	1%	68.26%
	BlockDrop	45M	-11%	91.31%	53M	-29%	67.39%
	Naive	34M	18%	91.27%	39M	5%	66.77%
	BranchyNet	33M	20%	91.37%	45M	-9%	67.00%
	S2DNAS	16M	61%	91.41%	25M	39%	67.29%
ResNet-56	Original	126M	-	93.03%	126M	-	71.32%
	LCCL	102M	19%	92.99%	106M	16%	70.33%
	BlockDrop	74M	41%	92.98%	129M	-2%	72.39%
	Naive	68M	46%	92.78%	108M	14%	71.58%
	BranchyNet	73M	42%	92.51%	120M	5%	71.22%
	S2DNAS	37M	71%	92.42%	62M	51%	71.20%
ResNet-110	Original	254M	-	93.57%	254M	-	73.55%
	LCCL	166M	35%	93.44%	210M	17%	72.72%
	BlockDrop	76M	70%	93.00%	153M	40%	73.70%
	Naive	158M	38%	93.13%	217M	15%	73.06%
	BranchyNet	147M	42%	93.33%	243M	5%	73.25%
	S2DNAS	76M	70%	93.39%	113M	56%	73.06%
VGG16-BN	Original	313M	-	93.72%	313M	-	72.93%
	LCCL	269M	14%	92.75%	264M	16%	70.46%
	Naive	185M	41%	93.34%	202M	36%	72.78%
	BranchyNet	162M	48%	93.39%	239M	24%	72.39%
	S2DNAS	66M	79%	93.51%	104M	67%	72.00%
MobileNetV2	Original	91M	-	93.89%	91M	-	74.21%
	LCCL	77M	15%	93.13%	73M	20%	71.11%
	Naive	38M	58%	91.90%	61M	33%	74.03%
	BranchyNet	35M	61%	91.76%	74M	18%	73.71%
	S2DNAS	25M	73%	92.25%	39M	57%	73.50%

As shown in Table 1, for most of the architectures and tasks, our method (denoted as S2DNAS in Table 1) can significantly reduce the computational cost with comparable accuracy with the original CNN model. As mentioned above, we use average FLOPs on the whole test dataset as the metric to measure the computational cost of a given CNN model. For ResNet-20 on CIFAR-10, S2DNAS has reduced the computation cost of the original net from 41M to 16M without the accuracy drop (even with a slight increase as shown in Table 1), which shows a relative cost reduction of 61%.

Our method also shows improvements over other methods for dynamic inference in terms of computational cost reduction. We have reproduced the previous works on these CNN models for comparison. We have also implemented a normal early exiting solution (marked as Naive in Table 1), i.e., directly adding prediction layers (i.e., global average pooling and fully-connected layers) at the intermediate layers of the original models. For example, for ResNet-20 on CIFAR-10, compared with BranchyNet [39], our method has achieved a slight accuracy improvement (from 91.37% to 91.41%) with more computational cost reduction.

One interesting observation is that some methods even cause an increase in computational cost. For example, BlockDrop boosts the FLOPs of the original net about 29%. We infer that this is caused by the controller with high computational cost introduced by BlockDrop in the inference process [45]. We also notice that some of the previous works can not be used for the network without residual connection. For instance, BlockDrop cannot be applied to VGG16-BN. In contrast, our method can generalize to CNN without residual connection. From Table 1, our method can reduce the computational cost of the original VGG16-BN net by 79% with a slight accuracy drop.

Comparison to MSDNet. As mentioned in the introduction section, there is a recent work that proposed a specialized CNN named MSDNet for dynamic inference. Since the method cannot directly be applied to general CNN models, thus for comparison with MSDNet [16], we use the DenseNet-like [17] model[10]. We then apply S2DNAS to it and generate the dynamic models. The results are plotted in Figure 5. The varying FLOPs metrics of the x-coordinate can be obtained by adjusting the thresholds of each classifier of the dynamic CNN models. As Figure 5 shows, in most cases, our method can achieve similar accuracy-computation trade-offs. In the case of CIFAR-10, MSDNet outperforms our method when FLOPs is relative to 15M. However, the superiority of MSDNet comes with the cost of manually designing the CNN architecture. In contrast, as Table 1 shows, our method can be applied to various general CNN models.

4.3   Discussion

Here, we present some discussions on our method for providing further insights.
Trade-off of Accuracy and Computational Cost. A key hyper-parameter of dynamic inference is the threshold setting $t = t_1, \cdots, t_s$, where s is the number of

---

[10] Using DenseNet-BC($d = 100$, $k = 8$) and doubling the growth rate after each transition layer
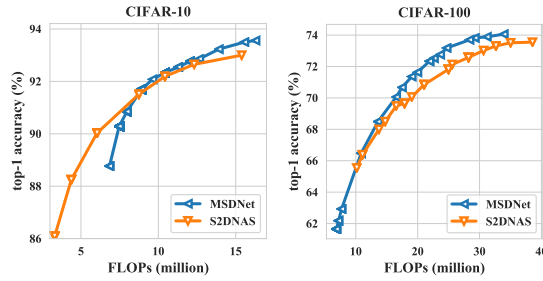
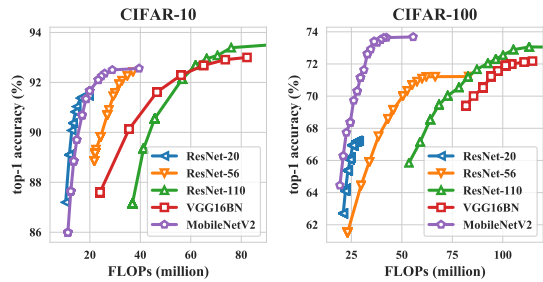**Fig. 5.** Comparison to MSDNet.

Fig. 6. Trade off accuracy with computational cost by adjusting thresholds of different stages.

stages. When the model is trained, different threshold settings lead to different trade-offs between the accuracy and the computational cost. To demonstrate how the threshold affects the final model performances, we conduct experiments with different thresholds and plot the results in Figure 6. All these results show the trend that the increase of computational cost leads to a performance boost. Thus, for practical use, we can set the threshold based on the computational budget of the given hardware device. Moreover, this property also helps to solve the anytime prediction task proposed in the prior work [16].

Difficulty Distribution of Test Dataset. The basic idea of our method is early exiting "easy" samples from the early stages. In this part, we give the statistics of all the samples in the test dataset (s=3, i.e., there are three stages in the trained model). As shown in Table 2, for ResNet-20 on CIFAR-10/100, the inference process of about 50% test samples exits from the first two stages. As a result, S2DNAS can considerably reduce the average computation cost. Further, we observe that the accuracy of the classifier in the first stage[11] is much higher than the classifier in later stages, which indicates that the classifier can easily classify those samples, i.e., those samples are "easy" samples. This

---

[11] Here, we only consider samples that exit from this stage.

Table 2. Accuracy and fractions of samples in test dataset that exit from each stage.

Dataset	Model	Stage1		Stage2		Stage3	
		Accuracy	Fractions	Accuracy	Fractions	Accuracy	Fractions
CIFAR-10	ResNet-20	98.44%	10.24%	98.89%	41.59%	83.45%	48.17%
	ResNet-56	98.25%	67.50%	89.72%	11.19%	75.36%	21.31%
	ResNet-110	98.43%	61.66%	93.22%	22.28%	74.28%	16.06%
	VGG-16BN	96.54%	87.29%	91.44%	2.22%	68.73%	10.49%
	MobileNetV2	98.62%	50.59%	94.04%	33.21%	68.70%	16.20%
CIFAR-100	ResNet-20	85.27%	58.72%	54.11%	20.68%	29.27%	20.60%
	ResNet-56	97.13%	22.27%	86.64%	29.86%	49.51%	47.87%
	ResNet-110	95.83%	28.04%	85.05%	28.90%	50.19%	43.06%
	VGG-16BN	97.21%	7.18%	90.08%	44.97%	51.20%	47.85%
	MobileNetV2	97.81%	8.68%	90.38%	45.84%	51.85%	45.48%

observation also validates the intuition ("easy" samples can be classified using fewer computations) pointed out by some recent works [2, 5, 16, 7].

5   Conclusion

In this paper, we present a general framework called S2DNAS, for transforming various static CNN models into multi-stage models to support dynamic inference. Empirically, our method can be applied to various CNN models to reduce the computational cost, without sacrificing model performance. In contrast to previous methods for dynamic inference, our method comes with two advantages: (1) With our method, we can obtain a dynamic model generated from an existing CNN model instead of manually re-designing a new CNN architecture. (2) The inference of the generated dynamic model does not introduce irregular computations or complex controllers. Thus the generated model can be easily deployed on various hardware devices using existing deep learning frameworks.

These advantages are appealing for deploying a given CNN model into hardware devices with limited computational resources. To be specific, we can first use S2DNAS to transform the given model into the dynamic one then deploy it on the hardware devices. Moreover, our method is orthogonal to previous pruning/quantization methods, which can further reduce the computational cost of the given CNN model. All these properties of our method imply a wide range of application scenarios where the efficient CNN inference is desired.

Acknowledgement

References

1. Konstantin Berestizshevsky and Guy Even. Dynamically sacrificing accuracy for reduced computation: Cascaded inference based on softmax confidence. In *Artificial Neural Networks and Machine Learning - ICANN 2019: Deep Learning - 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17-19, 2019, Proceedings, Part II*, pages 306–320, 2019.
2. Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 527–536, 2017.
3. Shijie Cao, Lingxiao Ma, Wencong Xiao, Chen Zhang, Yunxin Liu, Lintao Zhang, Lanshun Nie, and Zhi Yang. Seernet: Predicting convolutional neural network feature-map sparsity through low-bit quantization. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 11216–11225, 2019.
4. Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(4):834–848, 2018.
5. Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 1895–1903, 2017.
6. Michael Figurnov, Maxwell D. Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry P. Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 1790–1799, 2017.
7. Xitong Gao, Yiren Zhao, Lukasz Dudziak, Robert Mullins, and Cheng-Zhong Xu. Dynamic channel pruning: Feature boosting and suppression. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.
8. Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1737–1746, 2015.
9. Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.
10. Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1135–1143, 2015.
11. Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2980–2988, 2017.
12. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016.

13. Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: automl for model compression and acceleration on mobile devices. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*, pages 815–832, 2018.
14. Weizhe Hua, Christopher De Sa, Zhiru Zhang, and G. Edward Suh. Channel gating neural networks. *CoRR*, abs/1805.12549, 2018.
15. Weizhe Hua, Yuan Zhou, Christopher De Sa, Zhiru Zhang, and G. Edward Suh. Boosting the performance of CNN accelerators with dynamic fine-grained channel gating. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019.*, pages 139–150, 2019.
16. Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. Multi-scale dense networks for resource efficient image classification. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.
17. Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2261–2269, 2017.
18. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
19. Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.
20. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114, 2012.
21. Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pages 598–605, 1989.
22. Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.
23. Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. Improved techniques for training adaptive deep networks. *CoRR*, abs/1908.06294, 2019.
24. Xiaoxiao Li, Ziwei Liu, Ping Luo, Chen Change Loy, and Xiaoou Tang. Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6459–6468, 2017.
25. Darryl Dexu Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2849–2858, 2016.
26. Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L. Yuille, and Fei-Fei Li. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 82–92, 2019.

27. Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.
28. Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3675–3682, 2018.
29. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I*, pages 21–37, 2016.
30. Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. Conditional deep learning for energy-efficient and enhanced pattern recognition. In *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, pages 475–480, 2016.
31. Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, pages 525–542, 2016.
32. Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 779–788, 2016.
33. Mengye Ren, Andrei Pokrovsky, Bin Yang, and Raquel Urtasun. Sbnet: Sparse blocks network for fast inference. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 8711–8720, 2018.
34. Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 91–99, 2015.
35. Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4510–4520, 2018.
36. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.
37. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
38. Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 2820–2828. Computer Vision Foundation / IEEE, 2019.
39. Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *23rd International Conference*

on Pattern Recognition, ICPR 2016, Cancún, Mexico, December 4-8, 2016, pages 2464–2469, 2016.

40. Andreas Veit and Serge J. Belongie. Convolutional networks with adaptive inference graphs. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part I*, pages 3–18, 2018.

41. Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*, pages 420–436, 2018.

42. 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 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2074–2082, 2016.

43. Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 10734–10742, 2019.

44. Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 4820–4828, 2016.

45. Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogério Schmidt Feris. Blockdrop: Dynamic inference paths in residual networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 8817–8826, 2018.

46. Jiecao Yu, Andrew Lukefahr, David J. Palframan, Ganesh S. Dasika, Reetuparna Das, and Scott A. Mahlke. Scalpel: Customizing DNN pruning to the underlying hardware parallelism. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*, pages 548–560, 2017.

47. Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

48. Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 8697–8710, 2018.