BudgetNet: Neural Network Inference under Dynamic Budget Constraints

Anthony Chen, Sheng-De Wang
Department of Electrical Engineering, National Taiwan University, Taiwan
{r05921042, sdwang}@ntu.edu.tw

Abstract

Previous works have shown several approaches to obtain compact convolutional neural networks for mobile and embedded applications. However, because of the static nature of most convolutional neural networks, these approaches require training an individual model for each different budget constraints. In this paper, we introduce BudgetNet, a framework that dynamically regulates the computational cost of a given model during inference. Our framework directly selects components, blocks or paths, of a pre-trained model to evaluate a given image under an assigned budget constraint. We validate our proposed framework with ResNet and ResNeXt on CIFAR-10 and CIFAR-100 respectively. The results show that, with a single model, our method is more efficient when dealing with trade-offs between the computation cost and accuracy over a wide range of budget constraints. Based on the ResNeXt-29 (4×16d) model, our method can control the amount of computation from lowest 25 % towards original computation according to user's request while achieving 82.5% to 93.4% accuracy on CIFAR-10.

Introduction

Performing inference operations in deep convolutional neural networks (CNNs) on mobile and edge devices have become a primary trend in vision-based applications, such as face recognition (Hu et al. 2015) and autonomous vehicles (Bojarski et al. 2016).

In many practical applications, computational resources and latency demands for the same provided services on the same device could vary across time. A computational offloading scheme (Mtibaa et al. 2013) has been proposed to maximize the lifetime of the ensemble of mobile devices, and the works of (Gmach et al. 2007) involving workload trace data of a data center demonstrates time-varying service demands of enterprise applications. While compression techniques provide effective solutions to speed up inference operations, the characteristic of fixed computational costs during inference phase is unfavorable with dynamic circumstances. Some existing works propose to dynamically determine which subsets of the neural network architecture to engage in inference operations (Wu et al. 2018; Liu and Deng 2018; Teja Mullapudi et al. 2018). However, these works only change the inference path according to the properties of an input sample.

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

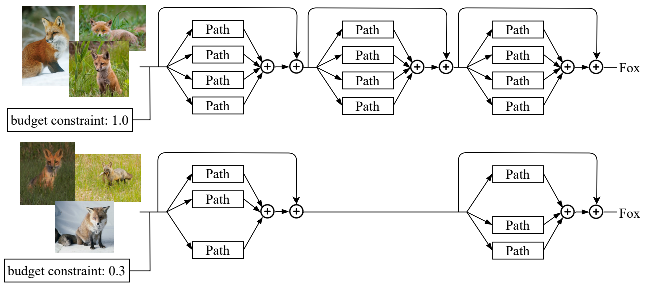

Figure 1: The fundamental concept of BudgetNet. A given model would use fewer components with lower budget constraint while tending to use the entire network with higher budget constraint during the inference phase.

In this paper, we introduce BudgetNet, a framework that dynamically regulates the computational cost of a given model during inference phase. In other words, users can adjust the components usage of a given model according to their demands, such as using 75% or 50% of maximum components inside the neural network. Fig. 1 shows the fundamental concept of BudgetNet.

Our contributions are listed as follows:

- We propose BudgetNet, a framework with reinforcement learning approach that can dynamically regulate the computational cost of a given model based on a target budget constraint during inference phase.
- BudgetNet can speedup inference operations of a given model in a particular interval of components usage without significant degradation in accuracy.
- We demonstrate the adaptability of BudgetNet by applying our framework to ResNeXt and ResNet. The extensive experimental results show that our method can perform more efficient trade-offs between computational cost and accuracy over a wide range of budget constraints with the pre-trained model.

Related Work

Model Compression

Building larger neural networks with more layers is a common trend in the most accurate model (Zoph et al. 2018). Because of limited storage space, memory size, and computational budgets, it has been increasingly difficult to deploy these models onto mobile and edge device. Many works on compressing and accelerating deep neural networks, such as parameter pruning (Li et al. 2016), quantization and binarizationis (Jacob et al. 2018), and low-rank factorization (Yu et al. 2017) are proposed to remove redundant structure and preserve informative weights. Unlike these compression approaches, which attempt to fix the amount of computation at all times, the objective of this work is to propose a computation-configurable network. Moreover, our approach is complementary to these model compression techniques.

Efficient Network Architecture

Apart from compressing complex models, many approaches directly train a compact neural network from scratch. SqueezeNet (Iandola et al. 2016) is a small CNN architecture for more efficient distributed training and more feasible embedded deployment. The main design strategies of SqueezeNet are to replace the size of original filters to smaller ones, decreasing the number of input channels and downsampling late in the network. MobileNets (Howard et al. 2017) is a class of efficient models for mobile and embedded vision applications. MobileNets factorize a standard convolution into a depthwise convolution and a point-wise convolution that drastically reduces computation and model size. Although these approaches provide hyper-parameters settings to train multiple models in varied size, the extra latency caused from frequently switching between models and the requirement of additional storage space is not ideal for the system in a dynamic context.

Dynamic Network

Several works are based on the idea of using a subset of the network specific for the input sample. HydraNet (Teja Mullapudi et al. 2018) contains multiple branches specialized for extracting features on similar classes, and a gate to decide which branches to execute at inference. Dynamic Deep Neural Networks (Li et al. 2018) achieves dynamic selective execution by augmenting a given network with control modules. Blockdrop (Wu et al. 2018) is a reinforcement approach to derive instance-specific inference. They propose the block dropping strategies that selectively choose residual blocks of a pre-trained ResNet to engage in inference operations. Although these works change the inference path based on the properties of an input sample, the computation cost of the model can not be manually adjusted. Incomplete dot product (IDP) operation (McDanel, Teerapittayanon, and Kung 2017) only used a subset of channels to adjust computational cost during forward propagation. URNet (Lee, Chang, and Kwak 2019) applied a Conditional Gating Module (CGM) to determine the usage of each residual block.

Our work differs from these approaches in the following ways. First, our work is based on reinforcement learning where as IDP modifies CNN training by adding a profile to provide an ordering for channels; URNet jointly trains CGM with ResNet through the scale loss. In addition, our work targets architecture-level dropping, where IDP focuses on channel-level dropping which is complementary to our approach. Also, our work provides a wider range of budget constraints in comparison to URNet under equivalent or better prediction accuracy.

The BudgetNet Model

Pre-trained Models with Branch Structure

In this work, we focus on how to dynamically regulate the computational cost of a given model based on target budget constraints during the inference phase. Therefore, we adopt existing CNNs as our pre-trained model rather than design a new network architecture. The work in (Veit, Wilber, and Belongie 2016) showed that removing individual modules from residual networks had minimal impact on classification error. This observation suggests that paths in a residual network do not strongly depend on each other. Accordingly, we apply ResNet as one of pre-trained models in our proposed framework by directly dropping selective residual block during inference phase.

ResNeXt, a variant of ResNet, exploits a split-transform-merge strategy in building blocks. This strategy introduces a hyper-parameter called cardinality which is the size of the set of transformations, or in other words, the number of independent paths. These independent paths can be considered as a kind of branch structure and is suitable for our requirements. All transformations functions (paths) share the same topology and the outputs of paths are summed up with element-wise addition. As a result, even if we neglect the outputs of some existing paths, the dimension of the building block's output would be unchanged and follow-up computations can still work properly. With this property, we can dynamically select appropriate paths of each building block to be executed during the inference phase. Adopting the pre-trained ResNeXt in our framework provides us with a more efficient way to utilize the network based on input samples and budget constraints, where we can not only drop the entire building block but also just mask partial paths of it.

Actor-Critic Method for Dynamic Inference

Reinforcement learning is a computational approach focusing on goal-directed learning from interaction. Our proposed framework is trained with an actor-critic method (Grondman et al. 2012; Konda and Tsitsiklis 2000; Mnih et al. 2016) where the actor decides which components of a given pre-trained model would engage in inference operations, and the critic predicts the return reward associated with the actor from the current state. Fig. 2 illustrates the overall structure of our proposed framework.

Policy Function and Value Function Unlike standard reinforcement learning, our policy π is based on the idea in (Wu et al. 2018), which outputs all actions at once to get an instantaneous reward. This is basically a one-step Markov Decision Process (MDP) that starts in some initial state s and

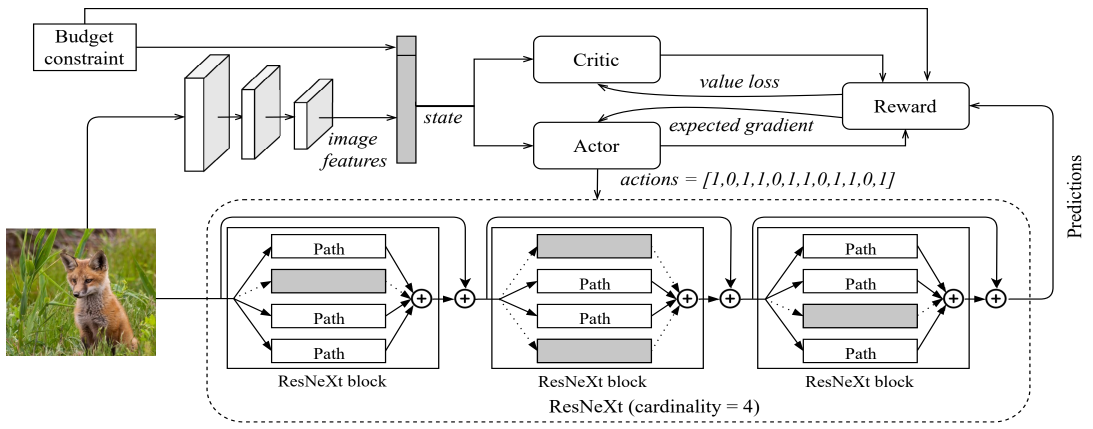

Figure 2: The overall structure of our proposed framework integrating with ResNeXt pre-trained model. We extract the patterns of a given image and then concatenate image features and target budget constraints into a single informative tensor. By feeding the tensor into the actor, we get a series of actions to decide which components of a given pre-trained model would engage in inference operations. The critic predicts the return reward associated with the actor from the same informative tensor.

that the episode terminates immediately after one time-step with reward r. There is no sequence of states and actions in this case, and the process is considered as a type of contextual bandit (Auer, Cesa-Bianchi, and Fischer 2002).

Given an input image x, a target budget constraint c, and a pre-trained model with K selectable components, we define a policy of selecting behavior as a K-dimensional Bernoulli distribution, and the policy function parameterized by θ can be derived as:

$$\pi_\theta(\mathrm{u}|\mathrm{x},\mathrm{c}) = \prod_{k=1}^{K} s_k^{u_k}(1-s_k)^{1-u_k} \tag{1}$$

$$\mathrm{s} = f_{\text{actor}}([f_{\text{extractor}}(\mathrm{x}),\mathrm{c}]) \tag{2}$$

where $f_{\text{extractor}}$ denotes the feature extractor which extracts the patterns of a given image x. Motivated by the conditional adversarial nets (Mirza and Osindero 2014), we believe that the value of target budget constraints can be viewed as a kind of auxiliary information. We directly concatenate target budget constraints and image features generated from $f_{\text{extractor}}$ into a single informative tensor. By feeding the tensor into the f_{actor}, we obtain a vector s in which element $\mathrm{s}_i \in [0,1]$ represents the probability of its corresponding component being executed in the original pre-trained model during the inference phase. An action $\mathrm{u}_i \in 0,1$ is determined based on s_i, where $\mathrm{u}_i = 0$ and $\mathrm{u}_i = 1$ respectively indicate dropping and keeping its corresponding components during the inference phase.

We adopt the state-value function of MDP to predict the return reward starting from state s with policy π. Using a neural network with parameters w, the state-value function can be parameterized as:

$$V_{\mathrm{w}}^{\pi}(\mathrm{x},\mathrm{c}) = f_{\text{critic}}([f_{\text{extractor}}(\mathrm{x}),\mathrm{c}]) \tag{3}$$

where f_{critic} is fed with the same informative tensor described in Eq. 2. State-value function generates the corresponding estimate of expected return. As illustrated in Fig. 2, the actor and the critic in our framework are two separate outputs of a single network, with shared initial layers constructing the feature representations.

Reward Function We design a reward function to describe how the actor ought to behave for different objectives. Our first reward function encourages the actor to select an appropriate amount of component based on the target budget constraints while maintaining a certain accuracy. The reward function is defined with:

$$R(\mathrm{u},\mathrm{c}) = \begin{cases} 1 - \sqrt{(|\frac{|\mathrm{u}|_0}{K} - \mathrm{c}|)} - \omega & \text{if correct} \\ -\gamma \times (1 + \sqrt{|(\frac{|\mathrm{u}|_0}{K} - \mathrm{c})|} - \omega) & \text{otherwise} \end{cases} \tag{4}$$

We measure the distance between the component usage of a given pre-trained model and the value of target budget constraints by calculating $\sqrt{|\frac{|\mathrm{u}|_0}{K} - \mathrm{c}|}$. When the network produces a correct prediction, we encourage the actor by giving a larger reward to the action using a closer percentage of components to the target budget constraints. On the other hand, if the network produces an incorrect prediction, we penalize the actor with γ multiplied by the weighted value based on the distance. That is, the action with improper component usage would get less reward or a larger penalty when the corresponding prediction is respectively correct or incorrect. Furthermore, to encourage the model to use fewer components, the reward would be decreased with additional penalty ω when the component usage is greater than the target budget constraint.

Optimization and Back-propagation In the training process, we would like to find the optimal policy parameters that maximize the following expected reward:

$$J(\theta) = \mathbb{E}_{\mathrm{u}\sim\pi_\theta}[R(\mathrm{u},\mathrm{c})] \tag{5}$$

To maximize Eq. 5, we adopt policy gradient algorithms (Sutton et al. 2000) to search for a local maximum in $J(\theta)$ by ascending the gradient of the policy parameterized with weights θ. Considering a simple one-step Markov Decision Process, we can estimate the policy gradient of our policy by using likelihood ratios (Glynn 1990) as:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\mathrm{u}\sim\pi_\theta}[R(\mathrm{u},\mathrm{c})\nabla_\theta \log \pi_\theta(\mathrm{u}|\mathrm{x},\mathrm{c})] \\ &= \mathbb{E}_{\mathrm{u}\sim\pi_\theta}[R(\mathrm{u},\mathrm{c})\nabla_\theta \log \prod_{k=1}^{K} \mathrm{s}_k^{\mathrm{u}_k}(1-\mathrm{s}_k)^{1-\mathrm{u}_k}]\end{aligned} \tag{6}$$

We obtain an unbiased estimate of the expected gradient in Eq. 6 with sample averages. To reduce the variance of the gradient estimate, we utilize the value function parameterized by **w** as the baseline and define the advantage function (Mnih et al. 2016) as:

$$A_{\mathrm{w}}(\mathrm{u},\mathrm{x},\mathrm{c}) = R(\mathrm{u},\mathrm{c}) - V_{\mathrm{w}}^{\pi}(\mathrm{x},\mathrm{c}) \tag{7}$$

where the advantage function describes the improvement compared to the expected reward of actions taken at that state. In other words, the positive and negative advantage values mean that the current action is a better or worse choice than the expected result. With the advantage function, Eq. 6 can be rewritten as:

$$\nabla_\theta J(\theta,\mathrm{w}) = \mathbb{E}_{\mathrm{u}\sim\pi_\theta}[A_{\mathrm{w}}(\mathrm{u},\mathrm{x},\mathrm{c})\nabla_\theta \log \prod_{k=1}^{K} \mathrm{s}_k^{\mathrm{u}_k}(1-\mathrm{s}_k)^{1-\mathrm{u}_k}]] \tag{8}$$

For the value function, we use smooth L1 loss to minimize the error between estimated reward and actual reward:

$$L_{\text{value}}(\mathrm{w}) = \text{smooth}_{L_1}(R(\mathrm{u},\mathrm{c}) - V_{\mathrm{w}}^{\pi}(\mathrm{x},\mathrm{c})), \tag{9}$$

where

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{10}$$

Training BudgetNet

Because the learning process of reinforcement learning is relatively unstable and sensitive to initial settings, we refer to the training schedule suggested in (Wu et al. 2018) and integrate the following training tips into our training process. We evaluate the return reward with Eq. 4 in the process of curriculum learning and joint fine-tuning.

Encouraging Exploration We apply a simple modification to the distribution s generated from f_{actor} to encourage exploration and to avoid saturation problems. The modified distribution s′ can be derived as:

$$\mathrm{s}' = \alpha^{1-\beta t} \cdot \mathrm{s} + (1 - \alpha^{1-\beta t}) \cdot (1 - \mathrm{s}) \tag{11}$$

where α bounds the distribution s, β is the decay constant and t is current epoch. The modified distribution is bounded in the range $1 - \alpha^{1-\beta t} \leq \mathrm{s}' \leq \alpha^{1-\beta t}$. The actor is further encouraged to explore diverse actions in the early training stage and would be less explorative as the number of epoch gradually increase. When $1 - \beta t = 0$, the modified distribution s′ is equal to original distribution s.

Curriculum Learning and Joint Fine-tuning Curriculum learning (Bengio et al. 2009) is a learning strategy where a model is trained from an initial set of easy samples that is expanded by gradually adding samples with increased difficulty level. In this spirit, we train our framework from easy to complex samples with two directions: 1) the number of controllable components, and 2) the range of budget constraints, in order to boost the learning process and guide our model to achieve a better local minimum.

The search space of our policy can be extremely large as the amount of controllable components rises. To improve the learning efficiency, we keep the first i components active, so that the actor only learns to control the last $K - i$ components. As i decreases, the amount of controllable components increases, and in the end the agent can control all the components to maximize the return reward. In addition to exploring from smaller search space in the beginning, we learn the policy from solving easier tasks first. A target budget constraint c is uniformly sampled in the range of $[c_{\text{min}}, 1.0]$ with each training image. Intuitively, it is much more difficult for a neural network to achieve similar accuracy by utilizing fewer components during prediction. Therefore, c_{min} is set to 1.0 in the beginning and we gradually expand the border to the lower bound ($c_{\text{min}} = 0.1$) as training epoch increases. After curriculum learning, the agent will be capable to determine which components in the pre-trained model is active for a given image and the target budget constraints in inference phase. While the pre-trained model is trained with fully utilizing every component, dropping a part of components in the original model would certainly influence accuracy. Therefore, we compensate for the accuracy degradation by jointly fine-tuning the pre-trained model with the agent.

Experiment

Experimental Setup

Datasets We evaluate BudgetNet on two competitive benchmarks: CIFAR10 and CIFAR100 (Krizhevsky 2009). The CIFAR datasets consist of 60000 color images with 32×32 pixels. There are 50000 training images and 10000 test images in 10 and 100 classes for CIFAR-10 and CIFAR-100, respectively. We adopt a basic data augmentation scheme (cropping and horizontal flipping) that is widely used for training image classification tasks. We also normalize the data by subtracting the mean from each pixel and then dividing the result by the standard deviation. For the following experiments, we evaluate the classification accuracy on the testing set.

Pre-trained Models We validate our framework with two pre-trained ResNeXt models on CIFAR. To be specific, ResNeXt-29 and ResNeXt-56 consist of 9 and 18 building blocks with cardinality = 4, which lead to 36 and 72 independent paths in the network, respectively. Table 1 shows the architecture of the above models in more detail. Furthermore, to compare the performance with URNet, we adopt the same ResNet-110 with 54 residual blocks model described in their work. These models are well pre-trained to the records on CIFAR before being integrated into our framework.

layer name	output size	ResNeXt-29 (4×16d)	ResNeXt-56 (4×16d)
conv1	32×32	3×3,16	
conv2_x	32×32	$\begin{bmatrix} 1\times 1, 16 \\ 3\times 3, 64, C=4 \\ 1\times 1, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 16 \\ 3\times 3, 64, C=4 \\ 1\times 1, 64 \end{bmatrix} \times 6$
conv3_x	16×16	$\begin{bmatrix} 1\times 1, 64 \\ 3\times 3, 128, C=4 \\ 1\times 1, 128 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 64 \\ 3\times 3, 128, C=4 \\ 1\times 1, 128 \end{bmatrix} \times 6$
conv4_x	8×8	$\begin{bmatrix} 1\times 1, 128 \\ 3\times 3, 256, C=4 \\ 1\times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 128 \\ 3\times 3, 256, C=4 \\ 1\times 1, 256 \end{bmatrix} \times 6$
	1×1	8×8 average pool 100-d fc, softmax	
# params.		1.13M	2.23M

Table 1: Pre-trained ResNeXt for CIFAR-100. The shape of building blocks are shown in brackets, and outside the brackets is the number of blocks stacked. Cardinality C represents the number of independent paths.

	CIFAR-10					CIFAR-100				
Target budget constraint c	0.2	0.4	0.6	0.8	1.0	0.2	0.4	0.6	0.8	1.0
URNet	92.2 18.08	93.3 20.86	93.7 32.02	93.7 44.37	93.6 52.19	70.7 28.10	71.5 28.57	72.4 32.00	73.0 44.61	72.8 49.41
BudgetNet (Ours)	80.7 4.28	89.7 7.75	92.6 13.18	93.1 17.53	93.3 20.24	69 22.2	71.5 23.51	73.5 29.6	73.8 36.84	74.1 41.39

Table 2: This figure compares our framework with URNet (Lee, Chang, and Kwak 2019) in ResNet-110 (54 blocks) on CIFAR datasets. The values shown in the cell are the classification accuracy (first row) and the amount of block usage (second row).

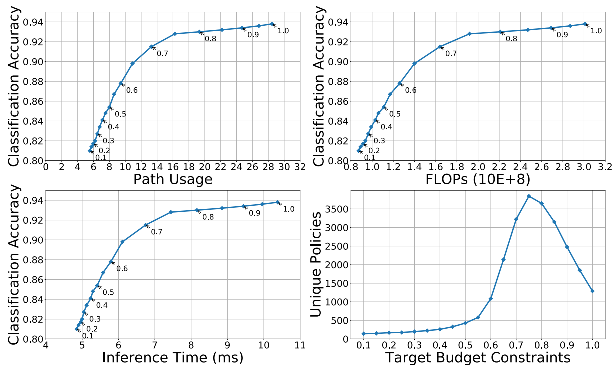

Figure 3: ResNeXt-29 (4×16d) on CIFAR 10.

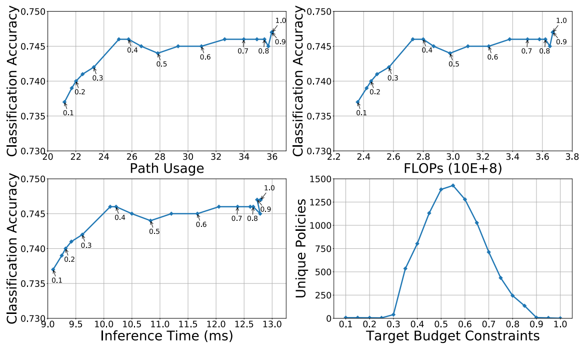

Figure 4: ResNeXt-29 (4×16d) on CIFAR 100.

Experiment Results

We report the results of the proposed framework integrated with ResNext-29 (4×16d) and ResNext-56 (4×16d) on CIFAR in Fig. 3 and Fig. 4; Each figure contains four different charts, three of them record the corresponding block usage, theoretical FLOPs, and the inference time of a given pre-trained model under different budget constraints. To make fair comparisons, each reported inference time is measured from the running results on the same NVIDIA GeForce GTX 1080 Ti GPU. While the last one records the amount of unique policies. We also compare our proposed framework with URNet, in which the authors introduced a similar concept of budget constraints called ”scale parameter”. Table 2 shows the proposed BudgetNet can have similar performance to the URNet but with less block usages, where the ResNet-110 pre-trained model (54 blocks) is tested under different target budget constraints.

Conclusion

In this paper, we present BudgetNet, a framework allowing users to regulate computational costs of a given model during the inference phase. We propose a reinforcement learning approach with actor-critic algorithms to train an agent by dynamically determining which subsets of the given neural network architecture are allowed to engage in inference operation based on the input image and target budget constraint. We deploy our framework on popular CNNs (ResNet and ResNeXt) and conduct extensive experiments on CIFAR. Experimental results show that our framework can efficiently trade off between the computation cost and accuracy over a wide range of budget constraints.

Our work can be extended in several directions. For example, we can integrate the existing feature extractor into convolutional modules of a pre-trained model to further reduce execution overhead. Our framework can also be applied from architecture-level selecting, blocks or paths, to channel-level selecting, filters, for further flexibility.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47:235–256.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, 41–48. ACM.

Bojarski, M.; Del Testa, D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L. D.; Monfort, M.; Muller, U.; Zhang, J.; et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Glynn, P. W. 1990. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM* 33(10):75–84.

Gmach, D.; Rolia, J.; Cherkasova, L.; and Kemper, A. 2007. Workload analysis and demand prediction of enterprise data center applications. In *2007 IEEE 10th International Symposium on Workload Characterization*, 171–180. IEEE.

Grondman, I.; Busoniu, L.; Lopes, G. A.; and Babuska, R. 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42(6):1291–1307.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hu, G.; Yang, Y.; Yi, D.; Kittler, J.; Christmas, W.; Li, S. Z.; and Hospedales, T. 2015. When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition. In *Proceedings of the IEEE international conference on computer vision workshops*, 142–150.

Iandola, F. N.; Han, S.; Moskewicz, M. W.; Ashraf, K.; Dally, W. J.; and Keutzer, K. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*.

Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; and Kalenichenko, D. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2704–2713.

Konda, V. R., and Tsitsiklis, J. N. 2000. Actor-critic algorithms. In *Advances in neural information processing systems*, 1008–1014.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Lee, S.; Chang, S.; and Kwak, N. 2019. Urnet : User-resizable residual networks with conditional gating module. *arXiv preprint arXiv:1901.04687*.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

Li, T.; Zhang, J.; Philip, S. Y.; Zhang, Y.; and Yan, Y. 2018. Deep dynamic network embedding for link prediction. *IEEE Access* 6:29219–29230.

Liu, L., and Deng, J. 2018. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

McDanel, B.; Teerapittayanon, S.; and Kung, H. 2017. Incomplete dot products for dynamic computation scaling in neural network inference. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 186–193. IEEE.

Mirza, M., and Osindero, S. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937.

Mtibaa, A.; Fahim, A.; Harras, K. A.; and Ammar, M. H. 2013. Towards resource sharing in mobile device clouds: Power balancing across mobile devices. In *ACM SIGCOMM Computer Communication Review*, volume 43, 51–56. ACM.

Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, 1057–1063.

Teja Mullapudi, R.; Mark, W. R.; Shazeer, N.; and Fatahalian, K. 2018. Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8080–8089.

Veit, A.; Wilber, M. J.; and Belongie, S. 2016. Residual networks behave like ensembles of relatively shallow networks. In *Advances in neural information processing systems*, 550–558.

Wu, Z.; Nagarajan, T.; Kumar, A.; Rennie, S.; Davis, L. S.; Grauman, K.; and Feris, R. 2018. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8817–8826.

Yu, X.; Liu, T.; Wang, X.; and Tao, D. 2017. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7370–7379.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 8697–8710.