AutoSlim: Towards One-Shot Architecture Search for Channel Numbers

Jiahui Yu Thomas Huang
University of Illinois at Urbana-Champaign

Abstract

*We study how to set channel numbers in a neural network to achieve better accuracy under constrained resources (*e.g.*, FLOPs, latency, memory footprint or model size). A simple and one-shot solution, named AutoSlim, is presented. Instead of training many network samples and searching with reinforcement learning, we train a single slimmable network to approximate the network accuracy of different channel configurations. We then iteratively evaluate the trained slimmable model and greedily slim the layer with minimal accuracy drop. By this single pass, we can obtain the optimized channel configurations under different resource constraints. We present experiments with MobileNet v1, MobileNet v2, ResNet-50 and RL-searched MNasNet on ImageNet classification. We show significant improvements over their default channel configurations. We also achieve better accuracy than recent channel pruning methods and neural architecture search methods.*

Notably, by setting optimized channel numbers, our AutoSlim-MobileNet-v2 at 305M FLOPs achieves ***74.2% top-1*** *accuracy, 2.4% better than default MobileNet-v2 (301M FLOPs), and even 0.2% better than RL-searched MNasNet (317M FLOPs). Our AutoSlim-ResNet-50 at 570M FLOPs, without depthwise convolutions, achieves* ***1.3% better*** *accuracy than MobileNet-v1 (569M FLOPs). Code and models will be available at:* https://github.com/JiahuiYu/slimmable_networks.

1. Introduction

The channel configuration (*a.k.a.* filter numbers or channel numbers) of a neural network plays a critical role in its affordability on resource constrained platforms, such as mobile phones, wearables and Internet of Things (IoT) devices. The most common constraints [1, 2, 3, 4, 5], *i.e.*, latency, FLOPs and runtime memory footprint, are all bound to the number of channels. For example, in a single convolution or fully-connected layer, the FLOPs (number of Multiply-Adds) increases linearly by the output channels. The memory footprint can also be reduced [6] by reducing the number of channels in bottleneck convolutions for most vision applications [6, 7, 8, 9].

Despite its importance, the number of channels has been chosen mostly based on heuristics. LeNet-5 [10] selected 6 channels in its first convolution layer, which is then projected to 16 channels after sub-sampling. AlexNet [11] adopted five convolutions with channels equal to 96, 256, 384, 384 and 256. A commonly used heuristic, the *"half size, double channel"* rule, was introduced in VGG nets [12], if not earlier. The rule is that when spatial size of feature map is halved, the number of filters is doubled. This heuristic has been more-or-less used in followup network architecture designs including ResNets [13, 14], Inception nets [15, 16, 17], MobileNets [6, 7] and networks for many vision applications [18, 19, 20, 21, 22]. Other heuristics have also been explored. For example, the pyramidal rule [23, 24] suggested to gradually increase the channels in all convolutions layer by layer, regardless of spatial size. Figure 1 visually summarizes these heuristics for setting channel numbers in a neural network.

Beyond the macro-level heuristics across entire network, recent works [6, 13, 24, 25, 26] have also digged into channel configuration for micro-level building blocks (a network building block is usually composed of several 1×1 and 3×3 convolutions). These micro-level heuristics have led to better speed-accuracy trade-offs. The first of its kind, *bottleneck residual block*, was introduced in ResNet [13]. It is composed of 1×1, 3×3, and 1×1 convolutions, where the 1×1 layers are responsible for reducing and then restoring dimensions, leaving the 3×3 layer a bottleneck ($4\times$ reduction). MobileNet v2 [6], however, argued that the bottleneck design is not efficient and proposed the *inverted residual block* where 1×1 layers are used for expanding feature first ($6\times$ expansion) and then projecting back after intermediate 3×3 depthwise convolution. Furthermore, MNasNet [25] and ProxylessNAS nets [26] included $3\times$ expansion version of *inverted residual block* into search space, and achieved even better accuracy under similar runtime latency.

Apart from these human-designed heuristics, efforts on automatically optimizing channel configuration have been made explicitly or implicitly. A recent work [27] suggested that many network pruning methods [1, 28, 29, 30, 31, 32]

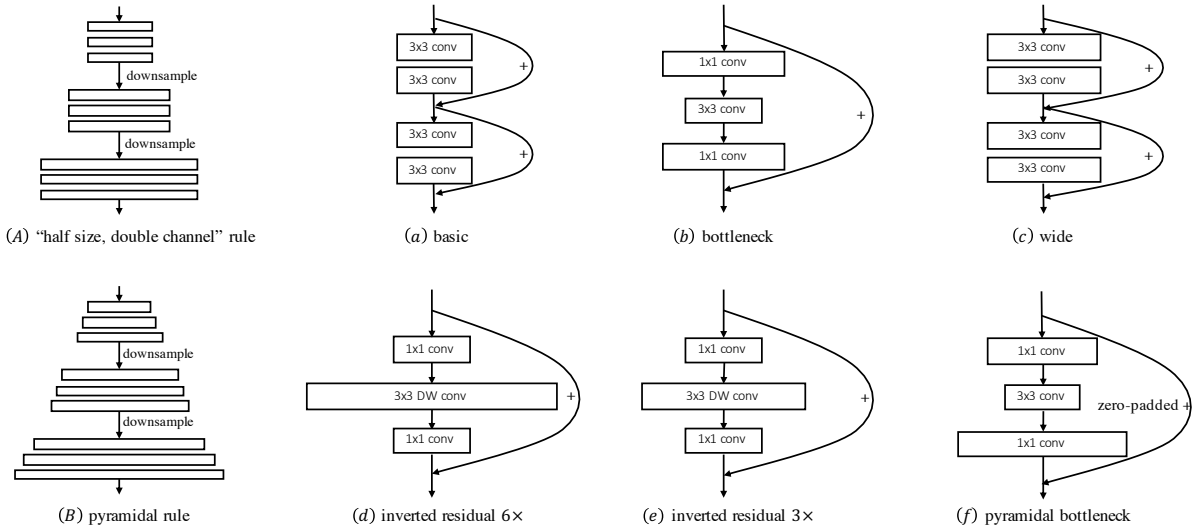

Figure 1. Various heuristics for setting channel numbers across entire network ($(A)-(B)$) [12, 23, 24], and inside network building blocks ($(a)-(f)$) [6, 13, 23, 24, 25, 26].

can be thought of as performing network architecture search for channel numbers. Liu *et al*. [27] showed that training these pruned architectures from scratch leads to similar or even better performance than fine-tuning and pruning from a large model. More recently, MNasNet [25] proposed to directly search network architectures, including filter sizes, using reinforcement learning algorithms [33, 34]. Although the search is performed on the factorized hierarchical search space, massive network samples and computational cost [25] are required for an optimized network architecture.

In this work, we study how to set channel numbers in a neural network to achieve better accuracy under constrained resources. To start, the first and the most brute-force approach came in mind is the exhaustive search: training all possible channel configurations of a deep neural network for full epochs (*e.g*., MobileNets [6, 7] are trained for approximately 480 epochs on ImageNet). Then we can simply select the best performers that are qualified for efficiency constraints. However, it is undoubtedly impractical since the cost of this brute-force approach is too high. For example, we consider a 8-layer convolutional networks and a search space limited to 10 candidates of channel numbers (*e.g*., 32, 64, ..., 320) for each layer. As a result, there are totally 10^8 candidate network architectures.

To address this challenge, we present a simple and one-shot solution *AutoSlim*. Our main idea lies in training a slimmable network [35] to approximate the network accuracy of different channel configurations. Yu *et al*. [35, 36] introduced slimmable networks that can run at arbitrary width with equally or even better performance than same architecture trained individually. Although the original motivation is to provide instant and adaptive accuracy-efficiency trade-offs, we find slimmable networks are especially suitable as benchmark performance estimators for several reasons: (1) Training slimmable models (using *the sandwich rule* [36]) is much faster than the brute-force approach. (2) A trained slimmable model can execute at arbitrary width, which can be used to approximate relative performance among different channel configurations. (3) The same trained slimmable model can be applied on search of optimal channels for different resource constraints.

In *AutoSlim*, we first train a slimmable model for a few epochs (*e.g*., 10% to 20% of full training epochs) to quickly get a benchmark performance estimator. We then iteratively evaluate the trained slimmable model and greedily slim the layer with minimal accuracy drop on validation set (for ImageNet, we randomly hold out $50K$ samples of training set as validation set). After this single pass, we can obtain the optimized channel configurations under different resource constraints (*e.g*., network FLOPs limited to 150M, 300M and 600M). Finally we train these optimized architectures individually or jointly (as a single slimmable network) for full training epochs. We experiment with various networks including MobileNet v1, MobileNet v2, ResNet-50 and RL-searched MNasNet on the challenging setting of 1000-class ImageNet classification. We compare our results with two baselines: (1) the default channel configuration of these networks, and (2) channel pruning methods on same network architectures [29, 30, 37, 38].

Our contributions are summarized as follows:

- We present the first one-shot approach on network architecture search for channel numbers with experiments on large-scale ImageNet classification.
- We demonstrate the importance of channel configuration in neural networks and the effectiveness of our approach on addressing this challenging problem.

- We achieve the state-of-the-art speed-accuracy trade-offs by setting the optimized channel configurations using *AutoSlim*.

2. Related Work

2.1. Architecture Search for Channel Numbers

In this part, we mainly discuss previous methods on automatic architecture search for channel numbers. Human-designed heuristics have been introduced in Section 1 and visually summarized in Figure 1.

Channel Pruning. Channel pruning (*a.k.a.*, network slimming) methods [1, 30, 39, 40, 41] aim at reducing effective channels of a large neural network to speedup its inference. Both training-based, inference-time and initialization-time pruning methods have been proposed [1, 30, 39, 40, 41, 42] in the literature. Here we selectively review two methods [1, 30]. He *et al*. [30] proposed an inference-time approach based on an iterative two-step algorithm: the LASSO based channel selection and the least square feature reconstruction. Liu *et al*. [1], on the other hand, trained neural networks with a ℓ_1 regularization on the scaling factors in batch normalization (BN) [43]. By pushing the factors towards zero, insignificant channels can be identified and removed. In a recent work [27], Liu *et al*. suggested that many network pruning methods [1, 28, 29, 30, 31, 32] can be thought of as performing network architecture search for channel numbers. In experiments, Liu *et al*. [27] showed that training these pruned architectures from scratch leads to similar or even better performance than iteratively fine-tuning and pruning a large model. Thus, Liu *et al*. [27] concluded that training a large, over-parameterized model is not necessary to obtain an efficient final model. In our work, we take channel pruning methods [29, 30, 37] as one of baselines.

Neural Architecture Search (NAS). Recently there has been a growing interest in automating the neural network architecture design [25, 26, 44, 45, 46, 47, 48, 49, 50, 51]. Significant improvements have been achieved by these automatically searched architectures in many vision and language tasks [47, 52]. However, most neural architecture search methods [44, 45, 46, 47, 48, 49, 50, 51] did not include channel configuration into search space, and instead applied human-designed heuristics. More recently, the RL-based searching algorithms are also applied to prune channels [37] or search for filter numbers [25] directly. He *et al*. proposed AutoML for Model Compression (AMC) [37] which leveraged reinforcement learning (deep deterministic policy gradient [53]) to provide the model compression policy. MNasNet [25] proposed to directly search network architectures, including filter sizes, for mobile devices. In the search, each sampled model is trained on 5 epochs using an aggressive learning rate schedule, and evaluated on a $50K$ validation set. In total, Tan *et al*. sampled about $8,000$ models during architecture search. Further, ProxylessNAS [26] proposed to directly learn the architectures for large-scale target tasks and target hardware platforms, based on DARTS [50]. For each residual block, ProxylessNAS [26] followed the channel configuration of MNasNet [25], while inside each block, the choices can be $\times 3$ or $\times 6$ version of *inverted residual blocks*. The memory consumption issue [26, 50] was addressed by binarizing the architecture parameters and forcing only one path to be active.

2.2. Slimmable Networks

Slimmable networks were firstly introduced in [35]. A general slimmable training algorithm and the switchable batch normalization were introduced to train a single neural network executable at different widths, permitting instant and adaptive accuracy-efficiency trade-offs at runtime. However, one drawback of the switchable batch normalization is that the width can only be chosen from a predefined widths set. The drawback was addressed in [36], where the authors introduced universally slimmable networks, extending slimmable networks to execute at arbitrary width, and generalizing to networks both with and without batch normalization layers. Meanwhile, two improved training techniques, *the sandwich rule* and *inplace distillation*, were proposed [36] to enhance training process and boost testing accuracy. Moreover, with the proposed methods, one can train nonuniform universally slimmable networks, where the width ratio is not uniformly applied to all layers. In other words, each layer in a nonuniform universally slimmable network can adjust its number of channels independently during inference. In this work, we simply refer to *nonuniform universally slimmable networks* as slimmable networks, if not explicitly noted. While the original motivation [35, 36] of slimmable networks is to provide instant and adaptive accuracy-efficiency trade-offs at runtime for different devices, we present an approach that uses slimmable networks for searching channel configurations of deep neural networks.

3. Network Slimming by Slimmable Networks

In this section, we first present an overview of our proposed approach for searching channel configuration of neural networks. We then discuss and analyze the difference of our approach compared with other baselines, *i.e*., network pruning methods and network architecture search methods. Afterwards we present each individual module in our proposed solution and discuss its non-trivial details.

3.1. Overview

The goal of channel configuration search is to optimize the number of channels in each layer, such that the network architecture with optimized channel configuration can

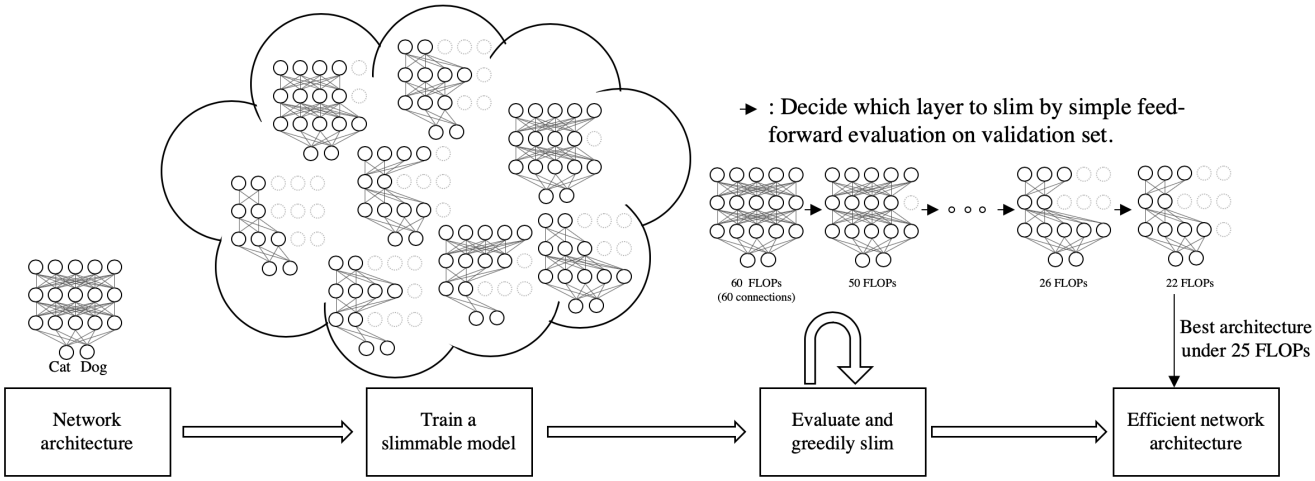

Figure 2. The flow diagram of our proposed approach *AutoSlim*.

achieve better accuracy under constrained resources. The constraints can be FLOPs, latency, memory footprint or model size. Our approach is conceptually simple, and it has two essential steps:

(1) Given a network architecture (*e.g.*, MobileNets, ResNets), we first train a slimmable model for a few epochs (*e.g.*, 10% to 20% of full training epochs). During the training, many different sub-networks with diverse channel configurations have been sampled and trained. Thus, after training one can directly sample its sub-network architectures for instant inference, using the correspondent computational graph and same trained weights.

(2) Next, we iteratively evaluate the trained slimmable model on the validation set. In each iteration, we decide which layer to slim by comparing their feed-forward evaluation accuracy on validation set. We greedily slim the layer with minimal accuracy drop, until reaching the efficiency constraints. No training is required in this step.

The flow diagram of our approach is shown in Figure 2. Our approach is also flexible for different resource constraints, since the FLOPs, latency, memory footprint and model size are all deterministic given a channel configuration and a runtime environment. By a single pass of greedy slimming in step (2), we can obtain the (FLOPs, latency, memory footprint, model size, accuracy) tuples of different channel configurations. It is noteworthy that the latency and accuracy are relative values, since the latency may be different across different hardware and the accuracy can be improved by training the network for full epochs. In the setting of optimizing channel numbers, we benefit from these relative values as performance estimators.

Discussion. We compare the flow diagram of our approach with the baselines, *i.e.*, network pruning methods and network architecture search methods.

Many network channel pruning methods [1, 4, 29, 32] follow a typical iterative training-pruning-finetuning pipeline, as shown in Figure 3. For example, Liu *et al.* [1] trained neural networks with a ℓ_1 regularization on the scaling factors in batch normalization (BN). After training, the method obtains channels in which many scaling factors are near zero for pruning. Pruning will temporarily lead to accuracy loss, thus the fine-tuning process and a repetitive multi-pass procedure are introduced for enhancement of final accuracy. Compared with our approach, a notable difference is that most network channel pruning methods are grounded on **the importance of trained weights**, thus the slimmed layer usually consists channels of discrete index (*e.g.*, the 4th, 7th, 9th channel are left as important channels while all others are pruned). In our approach, after slimmable training, the importance of the weight is *implicitly ranked by its index*. Thus our approach focuses more on **the importance of channel numbers**, and we always keep the lower-index channels (*e.g.*, all 1st to 3rd channels are left while 4th to 10th channels are slimmed in step (2)). We demonstrate the advantage of our approach by empirical evidences on ImageNet classification with various network

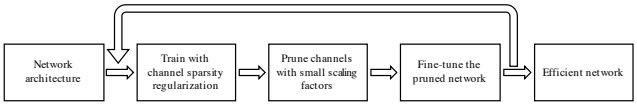

Figure 3. The flow diagram of network pruning methods [1].

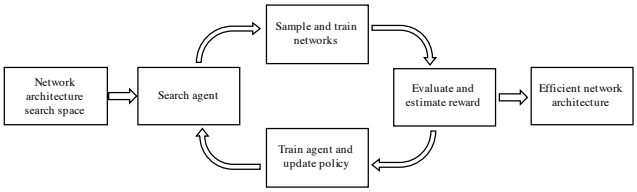

Figure 4. The flow diagram of network architecture search methods [25, 26, 47, 52].

architectures.

Network architecture search methods [25, 26, 47, 52] commonly consist of three major components: search space, search strategy, and performance estimation strategy. A typical pipeline is shown in Figure 4. First the search space is defined, based on which the search agent samples network architectures. The architecture is then passed to a performance estimator, which returns rewards (*e.g.*, predictive accuracy after training and/or network runtime latency) to the search agent. In the process, the search agent learns from the repetitive loop to design better network architectures. One major drawback of network architecture search methods is their high computational cost and time cost [46, 50]. Although recently differentiable architecture search methods [50, 54] were proposed, they cannot be applied on search of channel numbers directly. Most of them [50, 54] were still using human-designed heuristics for setting channel numbers, which may introduce human bias.

3.2. Training Slimmable Networks

Warmup. We warmup by a brief review of training techniques for slimmable networks. More details can be found in [35, 36]. Slimmable networks were firstly introduced and trained with switchable batch normalization [43], which employed individual BNs for different sub-networks. During training, features are normalized with current mini-batch mean and variance, thus a simple modification to switchable batch normalization is introduced in [36]: recalibrating BN statistics after training. With this simple modification, one can train universally slimmable networks [36] that can run with arbitrary channel numbers. Moreover, two improved training techniques *the sandwich rule* and *inplace distillation* were introduced to enhance training process and boost testing accuracy. We use all these techniques in training slimmable models by default.

Assumption. Our approach lies in the assumption that the slimmable model is a good accuracy estimator of individually trained models given same channel configuration. More specifically, we are interested in *the relative ranking of accuracy* among networks with different channel configurations. We use the instant inference accuracy of a slimmable model as the performance estimator. We note that assumptions and approximations commonly exist in other related methods. For example, in network channel pruning methods [1, 30], one may assume that weights with smaller norm are less informative and can be pruned, which may not be the case as shown in [39]. Recently the *Lottery Ticket Hypothesis* [42] was also introduced. In network architecture search methods [25, 26], one may believe the transferability among different datasets, accuracy approximations using aggressive learning rates and fewer training epochs, and approximation in runtime latency modeling.

The Search Space. The executable sub-networks in a slimmable model compose the search space of channel configurations given a network architecture. To train a slimmable model, we simply apply two width multipliers [7, 36] as the upper bound and lower bound of channel numbers. For example, for all mobile networks [6, 7, 25, 26], we train a slimmable model that can execute between $0.15\times$ and $1.5\times$. In each training iteration, we randomly and independently sample the number of channels in each layer. It is noteworthy that in residual networks, we first sample the channel number of residual identity pathway and then randomly and independently sample channel number inside each residual block. Moreover, we make all layers in a neural network slimmable, including the first convolution layer and last fully-connected layer. In each layer, we divide the channels into groups evenly (*e.g.*, 10 groups) to reduce the search space. In other words, during training or slimming, we sample or remove an entire group, instead of an individual channel. We note that even with channel grouping, the search space is still large.

We implement a distributed training framework with synchronized stochastic gradient descent (SGD) on PyTorch [55]. We set different random seeds in different processes such that each GPU samples diverse channel configurations in each SGD training step. All other techniques introduced in [35] and distributed training techniques introduced in [56] are used by default. All code will be released.

3.3. Greedy Slimming

After training a slimmable model, we evaluate it on the validation set (on ImageNet [58] we randomly hold out $50K$ images in training set as validation set). We start with the largest model (*e.g.*, $1.5\times$) and compare the network accuracy among the architectures where each layer is slimmed by one channel group. We then greedily slim the layer with minimal accuracy drop. During the iterative slimming, we obtain optimized channel configurations under different resource constraints. We stop until reaching the strictest constraint (*e.g.*, 50M FLOPs or 30ms CPU latency).

Large Batch Size. During greedy slimming, no training is involved. Thus we directly put the model in evaluation mode (no gradients are required), which enables us to use a larger batch size (for example during slimming we use mini-batch size 2048 for each GPU with totally 8 V100 GPUs). Large batch size brings two benefits. First, previous work [36] shows that BN statistics will be accurate if it is calibrated with the batch size larger than $2K$. Thus post-statistics of BN in our greedy slimming can be computed online without additional cost. Second, with large batch size we can simply use single feed-forward prediction accuracy as the performance estimator. In practice we find it speeds up greedy slimming and simplifies implementation without affecting final performance.

Training Optimized Networks. Similar to architecture

Table 1. ImageNet classification results with various network architectures. Blue indicates the network pruning methods [27, 29, 30, 37, 38], Cyan indicates the network architecture search methods [25, 47, 48, 57] and Red indicates our results using *AutoSlim*.

Group	Model	Parameters	Memory	CPU Latency	FLOPs	Top-1 Err. (gain)
200M FLOPs	ShuffleNet v1 1.0× [9]	1.8M	4.9M	46ms	138M	32.6
	ShuffleNet v2 1.0× [8]	-	-	-	146M	30.6
	MobileNet v1 0.5× [7]	1.3M	3.8M	33ms	150M	36.7
	MobileNet v2 0.75× [6]	2.6M	8.5M	71ms	209M	30.2
	AMC-MobileNet v2 [37]	2.3M	7.3M	68ms	211M	29.2 (1.0)
	MNasNet 0.75× [25]	3.1M	7.9M	65ms	216M	28.5
	AutoSlim-MobileNet v1	1.9M	4.2M	33ms	150M	32.1 (4.6)
	AutoSlim-MobileNet v2	4.1M	9.1M	70ms	207M	27.0 (3.2)
	AutoSlim-MNasNet	4.0M	7.5M	62ms	217M	26.8 (1.7)
300M FLOPs	ShuffleNet v1 1.5× [9]	3.4M	8.0M	60ms	292M	28.5
	ShuffleNet v2 1.5× [8]	-	-	-	299M	27.4
	MobileNet v1 0.75× [7]	2.6M	6.4M	48ms	325M	31.6
	MobileNet v2 1.0× [6]	3.5M	10.2M	81ms	300M	28.2
	NetAdapt-MobileNet v1 [38]	-	-	-	285M	29.9 (1.7)
	AMC-MobileNet v1 [37]	1.8M	5.6M	46ms	285M	29.5 (2.1)
	MNasNet 1.0× [25]	4.3M	9.8M	76ms	317M	26.0
	AutoSlim-MobileNet v1	4.0M	6.8M	43ms	325M	28.5 (3.1)
	AutoSlim-MobileNet v2	5.7M	10.9M	77ms	305M	25.8 (2.4)
	AutoSlim-MNasNet	6.0M	10.3M	71ms	315M	25.4 (0.6)
500M FLOPs	ShuffleNet v1 2.0× [9]	5.4M	11.6M	92ms	524M	26.3
	ShuffleNet v2 2.0× [8]	-	-	-	591M	25.1
	MobileNet v1 1.0× [7]	4.2M	9.3M	64ms	569M	29.1
	MobileNet v2 1.3× [6]	5.3M	14.3M	106ms	509M	25.6
	MNasNet 1.3× [25]	6.8M	14.2M	95ms	535M	24.5
	NASNet-A [47]	-	-	-	564M	26.0
	PNASNet-5 [48, 8]	-	-	-	588M	25.8
	Graph-HyperNetwork [57]	-	-	-	569M	27.0
	AutoSlim-MobileNet v1	4.6M	9.5M	66ms	572M	27.0 (2.1)
	AutoSlim-MobileNet v2	6.5M	14.8M	103ms	505M	24.6 (1.0)
	AutoSlim-MNasNet	8.3M	14.2M	95ms	532M	24.6 (-0.1)
Heavy Models	ResNet-50 [13]	25.5M	36.6M	197ms	4.1G	23.9
	ResNet-50 0.75× [13, 35]	14.7M	23.1M	133ms	2.3G	25.1
	ResNet-50 0.5× [13, 35]	6.8M	12.5M	81ms	1.1G	27.9
	ResNet-50 0.25× [13, 35]	1.9M	4.8M	44ms	278M	35.0
	He-ResNet-50 [30, 27]	-	-	-	≈2.0G	27.2
	ThiNet-ResNet-50 [29, 27]	-	-	-	≈2.9G	27.0
		-	-	-	≈2.1G	28.0
		-	-	-	≈1.2G	30.6
	AutoSlim-ResNet-50	23.1M	32.3M	165ms	3.0G	24.0
		20.6M	27.6M	133ms	2.0G	24.4
		13.3M	18.2M	91ms	1.0G	26.0
		7.4M	11.5M	69ms	570M	27.8

search methods, after the search, we train these optimized network architectures from scratch. By default we search for the network FLOPs at approximately 200M, 300M and 500M, and train a slimmable model.

4. Experiments

4.1. Main Results

Table 1 summarizes our results on ImageNet [58] classification with various network architectures including MobileNet v1 [7], MobileNet v2 [6], MNasNet [25], and one large model ResNet-50 [13]. We compare our results with their default channel configurations and recent channel pruning methods [29, 30, 37]. The top-1 errors of our baselines are from corresponding works [6, 7, 13, 25, 29, 30, 37]. To have a clear view, we divide the network architectures into four groups, namely, 200M FLOPs, 300M FLOPs, 500M FLOPs and heavy models (basically ResNet-50 based models). We evaluate their latency on same hardware environment with single-core CPU to ensure fairness. Device memory is reported as a summary of all feature maps and weights. We note that the memory footprint can be largely optimized by improving memory reusing and implementation of dedicated operators. For example, the *inverted residual block* can be optimized by splitting channels into groups and performing partial execution for multiple times [6]. For all network architectures we train 50 epochs with squeezed learning rate schedule to obtain a slimmable model for greedy slimming. After search, we train the optimized network architectures for full epochs (300 epochs with linearly decaying learning rate for mobile networks, 100 epochs with step learning rate schedule for ResNet-50 based models) with other training settings following previous works [6, 7, 8, 9, 13, 35, 36] (weight initialization, weight decay, data augmentation, training/testing image resolution, optimizer, hyper-parameters of batch normalization). We exclude the parameters and FLOPs of Batch Normalization layers [43] following common practice since they can be fused into convolution layers.

As shown in Table 1, our models have better top-1 accuracy compared with the default channel configuration of MobileNet v1, MobileNet v2 and ResNet-50 across different computational budgets. We even have improvements over RL-searched MNasNet [25], where the filter numbers are already included in its search space. Notably, by setting optimized channel numbers, our AutoSlim-MobileNet-v2 at 305M FLOPs achieves **74.2% top-1** accuracy, 2.4% better than default MobileNet-v2 (301M FLOPs), and even 0.2% better than RL-searched MNasNet (317M FLOPs). Our AutoSlim-ResNet-50 at 570M FLOPs, without depthwise convolutions, achieves **1.3% better** accuracy than MobileNet-v1 (569M FLOPs).

4.2. Visualization and Discussion

In this part, we visualize our optimized channel configurations and discuss some insights from the results.

Comparison with Default Channel Numbers. We first compare our results with default channels in MobileNet v2 [6]. We show the optimized number of channels (left) and the percentage compared with default channels (right) in Figure 5. Compared with default MobileNet v2, our optimized configuration has fewer channels in shallow layers and more channels in deep ones.

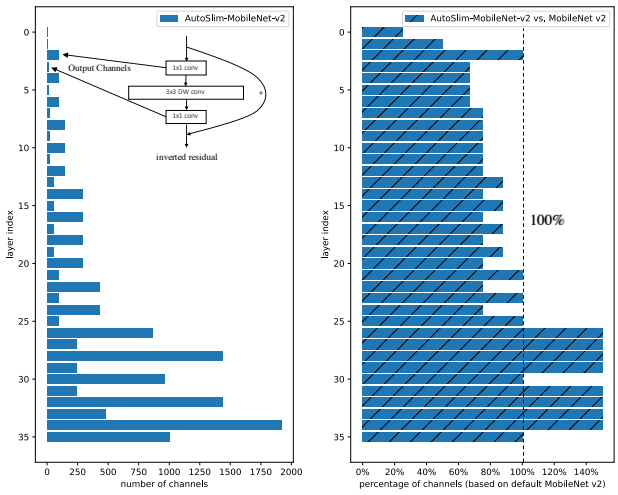

Figure 5. The optimized number of channels (left) and the percentage compared with default channels (right) of MobileNet v2. The channels of depthwise convolutions are ignored in the figure, since its output channels are always equal to the previous 1×1 convolution outputs.

Comparison with Width Multiplier Heuristic. Applying width multiplier [7], a global hyper-parameter across all layers, is a commonly used heuristic to trade off between model accuracy and efficiency [6, 7, 8, 9]. We search optimal channels at 207M, 305M and 505M FLOPs corresponding to MobileNet v2 $0.75\times$, $1.0\times$ and $1.3\times$. Figure 6 shows the pattern that under different budgets, *AutoSlim* applies different width scaling in each layer.

Comparison with Model Pruning Methods. Next, we compare our optimized channel configuration with model pruning method AMC [37]. In Figure 6, we show the number of channels in all layers of optimized MobileNet v2. We observe several characteristics of our optimized channel configurations. First, AutoSlim-MobileNet-v2 has much more channels in deep layers, especially for deep depthwise convolutions. For example, AutoSlim-MobileNet-v2 has 1920 channels in the second last layer, compared with 848 channels in AMC-MobileNet-v2. Second, AutoSlim-MobileNet-v2 has fewer channels in shallow layers. For example, AutoSlim-MobileNet-v2 has only 8 channels in first convolution layer, while AMC-MobileNet-v2 has 24 channels. It is noteworthy that although shallow layers have a small number of channels, the spatial size of feature maps is large. Thus overall these layers take up large computational overheads.

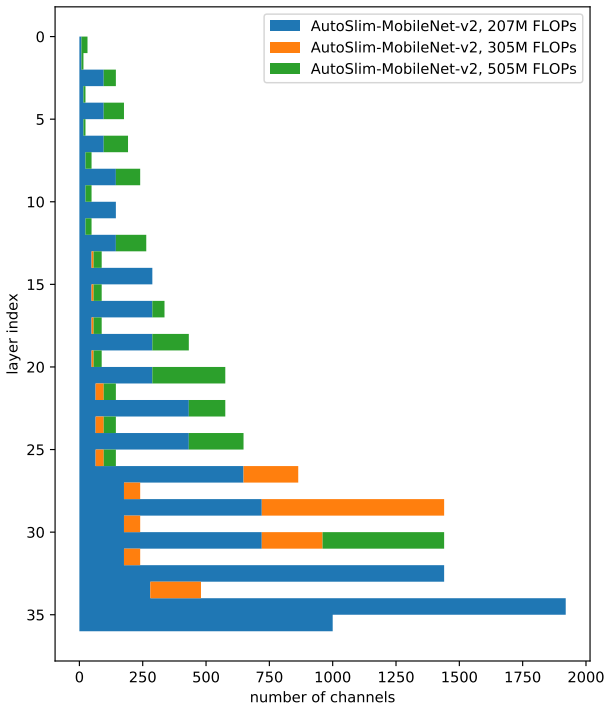

Figure 6. The channel configurations of AutoSlim-MobileNet-v2 at 207M, 305M and 505M FLOPs.

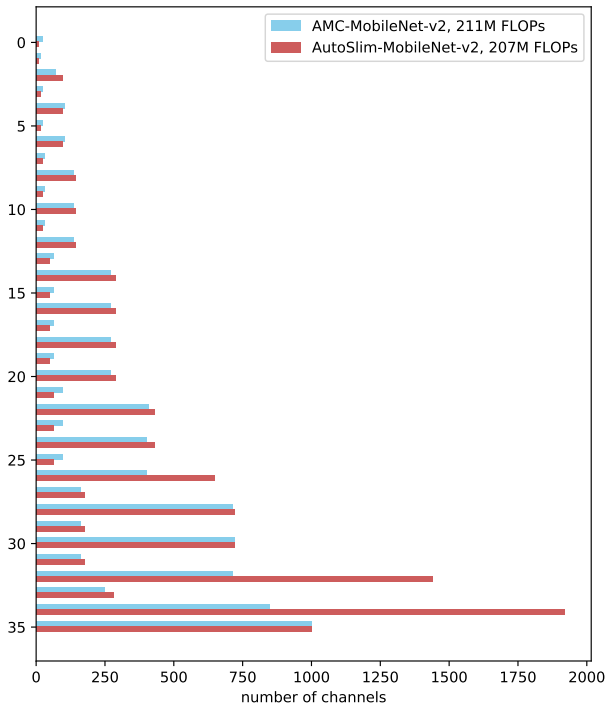

Figure 7. The channel configurations of AutoSlim-MobileNet-v2 compared with AMC-MobileNet-v2 [37].

Table 2. CIFAR10 classification results with default MobileNet v2 and AutoSlim-MobileNet-v2.

Model	Parameters	FLOPs	Top-1 Err.
MobileNet v2 1.0×	2.2M	88M	8.1
MobileNet v2 0.75×	1.3M	59M	8.6
MobileNet v2 0.5×	0.7M	28M	10.4
AutoSlim-MobileNet v2	1.5M	88M	6.8 (1.3)
AutoSlim-MobileNet v2	0.7M	59M	7.0 (1.6)
AutoSlim-MobileNet v2	0.3M	28M	8.0 (2.4)

Table 3. CIFAR10 results with AutoSlim-MobileNet-v2 searched on CIFAR10 or ImageNet.

Model	Search On	FLOPs	Top-1 Err.
MobileNet v2 0.75×	-	59M	8.6
AutoSlim-MobileNet v2	CIFAR10	59M	7.0 (1.6)
AutoSlim-MobileNet v2	ImageNet	63M	9.9 (-1.3)

4.3. CIFAR10 Experiments

In addition to ImageNet dataset, we also conduct experiments on CIFAR10 [59] dataset. We use same weight decay hyper-parameter, initial learning rate and learning rate schedule as ImageNet experiments. We note that these training settings may not be optimal for CIFAR10 dataset, nevertheless we report ablative study with same hyper-parameters and settings. We first report the performance of MobileNet v2 [6] with the default channel configurations. We then search with proposed *AutoSlim* to obtain optimized channel configurations at same FLOPs (we hold out $5K$ images from training set as validation set during the search). Finally we train the optimized architectures individually with same settings as the baselines. Table 2 shows that *AutoSlim* models have higher accuracy than baselines on CIFAR10 dataset.

We further study the transferability of the network architectures learned from ImageNet to CIFAR10 dataset, and compare it with the channel configuration searched on CIFAR10 directly. The results are shown in Table 3. It suggests that the optimized channel configuration on ImageNet cannot generalize to CIFAR10. Compared with the optimized architecture for ImageNet, we observed that the optimized architecture for CIFAR10 have much fewer channels in deep layers, which we guess may lead to better generalization on test set for small datasets like CIFAR10. It may also due to inconsistent image resolutions between ImageNet (224×224) and CIFAR10 (32×32).

5. Conclusion

We presented the first one-shot approach on network architecture search for channel numbers, with extensive experiments on large-scale ImageNet classification. Our proposed solution *AutoSlim* automates the design of efficient network architectures for resource constrained devices.

References

[1] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2755–2763. 1, 3, 4, 5

[2] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-scale dense networks for resource efficient image classification," *arXiv preprint arXiv:1703.09844*, 2017. 1

[3] X. Wang, F. Yu, Z.-Y. Dou, and J. E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," *arXiv preprint arXiv:1711.09485*, 2017. 1

[4] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015. 1, 4

[5] J. Yu, Y. Fan, J. Yang, N. Xu, X. Wang, and T. S. Huang, "Wide activation for efficient and accurate image super-resolution," *arXiv preprint arXiv:1808.08718*, 2018. 1

[6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *arXiv preprint arXiv:1801.04381*, 2018. 1, 2, 5, 6, 7, 8

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017. 1, 2, 5, 6, 7

[8] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 116–131. 1, 6, 7

[9] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," *arXiv preprint arXiv:1707.01083*, 2017. 1, 6, 7

[10] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. 1

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105. 1

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. 1, 2

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. 1, 2, 6, 7

[14] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE, 2017, pp. 5987–5995. 1

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9. 1

[16] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826. 1

[17] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. 1

[18] J. Yu, Y. Jiang, Z. Wang, Z. Cao, and T. Huang, "Unitbox: An advanced object detection network," in *Proceedings of the 24th ACM international conference on Multimedia*. ACM, 2016, pp. 516–520. 1

[19] Z. Zhang, S. Qiao, C. Xie, W. Shen, B. Wang, and A. L. Yuille, "Single-shot object detection with enriched semantics," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5813–5821. 1

[20] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, "Generative image inpainting with contextual attention," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5505–5514. 1

[21] Z. Shen, Z. Liu, J. Li, Y.-G. Jiang, Y. Chen, and X. Xue, "Dsod: Learning deeply supervised object detectors from scratch," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1919–1927. 1

[22] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, "Free-form image inpainting with gated convolution," *arXiv preprint arXiv:1806.03589*, 2018. 1

[23] D. Han, J. Kim, and J. Kim, "Deep pyramidal residual networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5927–5935. 1, 2

[24] K. Zhang, L. Guo, C. Gao, and Z. Zhao, "Pyramidal ror for image classification," *Cluster Computing*, pp. 1–11, 2017. 1, 2

[25] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," *arXiv preprint arXiv:1807.11626*, 2018. 1, 2, 3, 4, 5, 6, 7

[26] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018. 1, 2, 3, 4, 5

[27] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018. 1, 2, 3, 6

[28] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016. 1, 3

[29] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," *arXiv preprint arXiv:1707.06342*, 2017. 1, 2, 3, 4, 6, 7

[30] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1398–1406. 1, 2, 3, 5, 6, 7

[31] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 304–320. 1, 3

[32] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143. 1, 3, 4

[33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017. 2

[34] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, M. Riedmiller *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017. 2

[35] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," *arXiv preprint arXiv:1812.08928*, 2018. 2, 3, 5, 6, 7

[36] J. Yu and T. Huang, "Universally slimmable networks and improved training techniques," *arXiv preprint arXiv:1903.05134*, 2019. 2, 3, 5, 7

[37] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 784–800. 2, 3, 6, 7, 8

[38] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 285–300. 2, 6

[39] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," *arXiv preprint arXiv:1802.00124*, 2018. 3, 5

[40] Q. Huang, K. Zhou, S. You, and U. Neumann, "Learning to prune filters in convolutional neural networks," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 709–718. 3

[41] N. Lee, T. Ajanthan, and P. H. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *arXiv preprint arXiv:1810.02340*, 2018. 3

[42] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Training pruned neural networks," *CoRR*, vol. abs/1803.03635, 2018. [Online]. Available: http://arxiv.org/abs/1803.03635 3, 5

[43] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015. 3, 5, 7

[44] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *arXiv preprint arXiv:1808.05377*, 2018. 3

[45] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *International Conference on Machine Learning*, 2018, pp. 549–558. 3

[46] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018. 3, 5

[47] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710. 3, 4, 6

[48] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 19–34. 3, 6

[49] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," *arXiv preprint arXiv:1711.00436*, 2017. 3

[50] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018. 3, 5

[51] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," *arXiv preprint arXiv:1708.05344*, 2017. 3

[52] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016. 3, 4

[53] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015. 3

[54] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Advances in Neural Information Processing Systems*, 2018, pp. 7827–7838. 5

[55] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017. 5

[56] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017. 5

[57] C. Zhang, M. Ren, and R. Urtasun, "Graph hypernetworks for neural architecture search," *arXiv preprint arXiv:1810.05749*, 2018. 6

[58] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. Ieee, 2009, pp. 248–255. 5, 7

[59] A. Krizhevsky, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009. 8