A GRADIENT-BASED ARCHITECTURE HYPERPARAMETER OPTIMIZATION APPROACH

Anonymous authors
Paper under double-blind review

ABSTRACT

Network hyperparameters, such as network depth, layer-wise channel numbers, and input image resolution, are crucial for designing high-performance neural network architectures under resource limited scenarios. Previous solutions either optimize these hyperparameters with customized algorithms, or enumerate the hyperparameters with confined choices. Those methods are laborious and cumbersome to obtain a good solution. In this work, we propose a gradient-based approach to optimize these parameters in an efficient and unified manner, based on the observation that these parameters are consecutive and network performance changes continuously with them. Specifically, natural evolutionary strategy (NES) is used to approximate the gradient of the non-differentiable architecture hyperparameters and we incorporate it into the gradient descent framework for joint optimizing the weights and architecture hyperparameters. Compared to the state-of-the-art method, ChamNet, our method achieves higher accuracy with much fewer optimization time cost. Our method easily surpasses state-of-the-art methods and achieves up to 9.1%/6.1% accuracy enhancement than compact models MobileNet v1/v2.

1 INTRODUCTION

Neural Architecture Search (NAS) aims at automatically designing effective network architectures. Most NAS algorithms focus at searching the optimal operators in a network layer/block (Cai et al., 2018; Wu et al., 2018) or their connection patterns (Liu et al., 2018a). Note that these design choices are *categorical* and *unordered*.

There are other important design choices in the neutral network, such as the layer-wise channel numbers, the input image resolution and the network depth, called architecture hyperparameters. Note that these design choices are fundamentally different because they are *integers* and are *ordered*.

These architecture hyperparameters are not independent. Intuitively, when image resolution is high, the network should be deep such that the receptive field is large enough. Also the network should have more channels to model the fine-grained image details (Tan & Le, 2019). However, optimizing these hyperparameters jointly is challenging for existing approaches. There is not yet an effective yet efficient solution. The challenges are three folds:

(1) Many existing algorithms are designed for optimizing a specific dimension of the architecture hyperparameter based on its unique property. Such algorithms cannot be generalized for other dimensions. For example, to decide optimal channel numbers, it is common to use the L1-norm as the importance indicator so that less important channels are pruned. This approach is not applicable for image spatial resolution.

(2) The search space is combinatorially large. For example, in a typical case, MobileNet v2 has 19 blocks. Channel number in each block is chosen from 32 to 128. The input image resolution is chosen from 96 to 224. The depth is chosen from 10 to 19. This search space has $96^{19} \times 128 \times 10 \approx 6^{40}$ architectures. It is 10^{22} times larger than a search space for the categorized and unordered objectives, for example, 6^{18} choices in (Wu et al., 2018)). This renders the existing NAS approach infeasible for hyper parameter search.

(3) Existing algorithms for architecture hyperparameter optimization are very inefficient. The state-of-the-art algorithm, ChamNet (Dai et al., 2018), models the architecture hyperparameter optimiza-

tion as a black-box regression task of predicting the hyperparameter-accuracy curve. For regression training, hundreds of architectures are trained from scratch. This is very slow. EfficientNet (Tan & Le, 2019) use grid search to find the optimal ratio of depth/width/resolution, which also requires to sample a great number of networks and train them separately.

This work proposes a simple and efficient method. We observe that architecture hyperparameters are ordered integers. Our target loss function typically changes continuously with such hyperparameters. This naturally motivates a gradient descent based paradigm. As the loss function is non-differentiable with respect to the architecture hyperparameters, we use natural evolutionary strategy (NES) to approximate the gradient of the architecture hyperparameters. We note that gradient-based methods are used in previous NAS approaches (Liu et al., 2018a; Wu et al., 2018; Xie et al., 2018). These methods are for categorical and unordered designed choices. They relax the bi-level mask to continuous changes in [0,1]. They are not applicable to the architecture hyperparameter optimization problem.

To verify the effectiveness of our proposed method, we apply it on MobileNets (Howard et al., 2017; Sandler et al., 2018) and ResNet (He et al., 2016). We achieve up to $9.1\%/6.3\%$ higher accuracy than MobileNet V1/V2 and up to 1.3% accuracy enhancement than ResNet. More accuracy gain can be obtained when optimizing the architecture hyperparameters targeting at the latency constraint, for which the proposed method can discover and utilize the hardware characteristic. At the same latency, our accuracy is 3.9%-16.2% higher than MobileNet V1, and 1.0%-13.1% higher than MobileNet V2. Compared with state-of-the-art hyperparameter learning algorithm (Dai et al., 2018), our method achieved higher or comparable results and is much more efficient.

Our contributions are three-folds: 1) We propose a novel gradient based approach to jointly optimize the architecture hyperparameters in a unified manner. 2) We adopt natural evolution strategy to approximate the gradient of the non-differentiable architecture hyperparameters. 3) We verify the effectiveness and efficiency of proposed method with state-of-the-art results in various network structures.

2 Related Works

Neural Architecture Search Neural architecture search (NAS) algorithms achieve state-of-the-art results (Zoph et al., 2018; Tan et al., 2018; Wu et al., 2018; Cai et al., 2018). But the search spaces of these algorithms are mainly limited to the discrete choices such as the operations in each layer (Wu et al., 2018; Stamoulis et al., 2019; Guo et al., 2019; Cai et al., 2018), or the connection patterns (Liu et al., 2018a; Xie et al., 2019), which are categorized unordered choices. The algorithm designed for these search space can hardly applied to the architecture hyperparameter optimization.

Pruning and AutoML Pruning could be viewed as architecture hyperparameter optimization, with respect to the layer-wise channel numbers (Liu et al., 2018b). Traditional pruning methods (Hu et al., 2016; Ding et al., 2019b;a; 2018; Li et al., 2016; Mariet & Sra, 2016; Yu et al., 2018; Yu & Huang, 2019; Huang & Wang, 2018) and AutoML-based pruning (He et al., 2018b; Yang et al., 2018; Liu et al., 2019) are all very effective for optimizing network accuracy under resource constraints. But solely dealing with the channel dimension limits the compression ratio as well as the accuracy upper-bound of the compressed network.

Hyperparameter Optimization General Hyperparameter Optimization is developed for tens of years (Bergstra et al., 2011). The optimization methods includes sequential model-based global optimization (SMBO) (Hutter et al., 2011), grid search, random search (Bergstra & Bengio, 2012), Bayesian optimization (Snoek et al., 2012) and reinforcement learning (Li, 2017).

Architecture hyperparameter optimization is a sub-class of the hyperparameter optimization. ChamNet and EfficientNet (Dai et al., 2018; Tan & Le, 2019) are two famous state-of-the-art papers. Despite their high accuracy, ChamNet is costly in obtaining the evaluation accuracy while the EfficientNet uses a grid search and has a confined search space.

Gradient Estimation and Evolutionary Strategy Gradient estimation is a widely-used technique (Koutnik et al., 2010; Fu, 2015; Li & Turner, 2017; Glasserman & Ho, 1991; Shi et al., 2018; Buesing et al., 2016; Maclaurin et al., 2015). A review of gradient estimation can be found in (Fu, 2006). Several works apply gradient approximation to the neural networks (Andrychowicz

et al., 2016; Schulman et al., 2015). But they are targeting at the ordinary weight parameters instead of architecture hyperparameters, which is different from our task.

Evolutionary strategy (ES) is a class of black box optimization algorithms (Rechenberg, 1994), which includes evolution algorithm(EA) (Qin et al., 2008; Qin & Suganthan, 2005; Liu & Lampinen, 2005; Farhi et al., 2001; Mallipeddi et al., 2011), covariance matrix adaptation evolution strategy(CMA-ES) (Hansen, 2016), etc. A comprehensive introduction of evolutionary strategy can be found in (Beyer & Schwefel, 2002). In this work, the evolutionary strategy we used belongs to the natural evolution strategies(NES) (Wierstra et al., 2008; 2014; Sun et al., 2009; Glasmachers et al., 2010; Schaul et al., 2011; Sehnke et al., 2010), which iteratively update the continuous parameters of a search distribution by following the natural gradient towards higher expected fitness. Our method is mostly related to (Salimans et al., 2017).

3 METHODOLOGY

We formulate the architecture hyperparameter optimization problem as

$$\min_{\mathcal{H}} \min_{\mathcal{W}} \mathcal{L}(\mathcal{H}; \mathcal{W}), \tag{1}$$

where we jointly optimize the hyperparameters $\mathcal{H}$ of the channel, spatial and depth dimension for a backbone architecture with the goal of minimizing the loss $\mathcal{L}$ when the weights $\mathcal{W}$ corresponding to $\mathcal{H}$ are trained. In this task, the loss is defined as,

$$\mathcal{L} = \mathcal{L}_c + \rho||\mathcal{R} - \mathcal{R}_t||^2, \tag{2}$$

where $\mathcal{L}_c$ denotes the classification loss on the evaluation data split, ρ denotes the regularization factor, the $\mathcal{R}$ and $\mathcal{R}_t$ denote the resource consumption of current model and the target resource constraint, respectively.

In this optimization problem, there are two challenges that make the previous solutions unsuitable or less efficient for this task: (1) combinatorially large search space; (2) resource intensity in obtaining the evaluation accuracy for various architectures. We explain these challenges and proposed our solution in Section 3.1 and 3.2 respectively, in Section 3.3 we explain the pipeline of joint optimizing the architecture hyperparameters and the weight parameters in the neural network.

3.1 GRADIENT-BASED METHOD FOR ARCHITECTURE HYPERPARAMETER OPTIMIZATION IN LARGE SEARCH SPACE

The search space for jointly optimizing the architecture hyperparameter in three dimensions is combinatorially large, which makes the algorithms designed for the categorized and unordered objectives ineffective. To tackle with that, we proposed a gradient-based method customized for the ordered architecture hyperparameters. Different from the existing gradient-based methods that relax the bi-level choice of different operations, proposed gradient-based method is built on top of an observation that the architecture hyperparameters are ordered integers and the loss of the neural network changes continuously with respect to the architecture hyperparameters. Since the algorithm is non-differentiable with respect to the architecture hyperparameters, we adopt natural evolutionary strategy (NES) for gradient approximation.

3.1.1 A REVIEW OF NATURAL EVOLUTIONARY STRATEGY

Natural Evolutionary strategy (NES) is a recent family of black-box optimization algorithms that use the natural gradient to update a parameterized search distribution in the direction of higher expected fitness (Wierstra et al., 2008; 2014).

In NES, to measure the gradient of the continuous vector input x, random gaussian noises n with the deviation of σ are added to the vector:

$$x' = x + n,\ n \sim N(0, \sigma), \tag{3}$$

Each noise n will result in a change in the output:

$$\Delta\mathcal{F} = \mathcal{F}(x') - \mathcal{F}(x), \tag{4}$$

where $\mathcal{F}$ is the objective function. By weighting the noise directions with the change in output function and taking an average, the gradient $\mathcal{G}$ that minimizes the loss function can be approximated:

$$\mathcal{G} \approx \frac{1}{\sigma \mathcal{M}} \sum_{i=0}^{\mathcal{M}} \Delta \mathcal{F} n_i, \tag{5}$$

where $\mathcal{M}$ denotes the number of noises added to the input vector.

In previous works, the NES is applied to solving various black-box optimization problems (Wierstra et al., 2008; 2014; Sun et al., 2009; Glasmachers et al., 2010; Schaul et al., 2011; Sehnke et al., 2010) or used as an alternative to the reinforcement learning in playing video games (Salimans et al., 2017). To the best of our knowledge, we are the first to apply NES in neural architecture hyperparameter optimization and prove it to be effective.

3.1.2 NES FOR ARCHITECTURE HYPERPARAMETER OPTIMIZATION

Different from the unordered operation search, where the operation choice 1 and choice 2 can arbitrarily change their index, the architecture hyperparameters are ordered continuous integers in all the three dimensions. This ordered property implies a learnable direction information between choices, which inspires us to use a local gradient to find the direction for optimizing the architecture hyperparameters.

In this work, we apply the natural evolutionary strategy for approximating the architecture hyperparameters, in which, the change in the output loss $\mathcal{L}$ can be obtained by varying the hyperparameters $\mathcal{H}$ randomly in different directions n_i,

$$\Delta \mathcal{L}_i = \mathcal{L}(\lfloor \mathcal{H} + n_i \rfloor, \mathcal{W}) - \mathcal{L}(\mathcal{H}, \mathcal{W}), \;\; n_i \sim N(0, \sigma). \tag{6}$$

Note that if $\mathcal{H} + n_i$ are not integers, we simply round them down to integers, with $\lfloor \rfloor$. Then by weighting different variation directions with the changing in loss, we can approximate the gradient of architecture hyperparameters towards the loss descending direction,

$$\mathcal{G} \approx \frac{1}{\sigma \mathcal{M}} \sum_{i=0}^{\mathcal{M}} \Delta \mathcal{L}_i n_i. \tag{7}$$

Natural evolutionary strategy (NES) is chosen for architecture hyperparameter optimization because it can utilize the continuity information and use a gaussian kernel to increase the stability in the gradient approximation. We know that the gradient information in a tiny local region in the neural network can be highly unstable. In NES, the gaussian distributed noise has a large probability in the near local as well as a small probability in the further area. Thus, by adding the Gaussian noise to the architecture hyperparameters for gradient estimation, the estimation process can combine both adjacent information as well as the information in further areas. This property helps to overcome the instability of local gradient estimation in the neural network.

3.2 WEIGHT SHARING FOR OVERCOMING THE RESOURCE INTENSITY IN OBTAINING THE EVALUATION LOSS

In order to make the gradient-based method with respect to the architecture hyperparameters function well, we need to obtain the evaluation loss of the network corresponding to each architecture hyperparameters on the validation dataset, which is splitted from the training dataset. Since the evaluation loss is a function of weights and architecture hyperparameters, to get the reliable loss prediction of different architecture hyperparameters, the weights of the corresponding architectures need to be trained. Using random untrained weights will result in arbitrary network loss which can hardly provide any useful information in hyperparameter updating. However, training the weights for each hyperparameter-defined architecture separately could be too resource-consuming.

Inspired by one-shot architecture search (Guo et al., 2019; Liu et al., 2019; Bender et al., 2018), we proposed to use the weight sharing mechanism in training weights for different architectures. Different from the layer-wise weight-sharing mechanism designed for operations, in which each operation has its own weights and different networks shares the weights when they choose the same operation

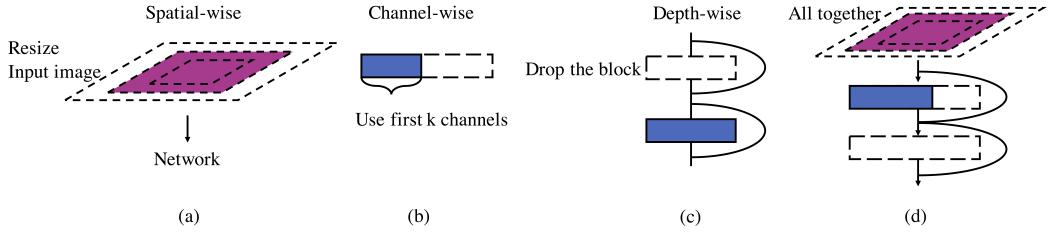

Figure 1: The designed weight sharing mechanism for each dimension.

in a layer, our weight sharing method is designed for continuous integer architecture hyperparameters. Here, we explain the weight sharing method in spatial, channel and depth dimension.

For architectures with different input image resolutions, we share the weights in the entire network and just rescale the input image (Figure 1 (a)), since different input image size only affect the feature map size and do not require any change in the weight kernel size. In the channel dimension, a weight matrix of the maximum number of channels is stored. Given the hyperparameter specifying the number of channels in the particular layer, we crop the weights for first k channels and remaining weights are untouched (Figure 1 (b)). For the depth dimension, we can drop the block to decrease the depth and use the shortcut to propagate the identity mapping only (Figure 1 (c)). Combining these weight sharing technique together, we can construct the network for weight training in architecture hyperparameter optimization (Figure 1 (d)). The weight sharing technique enables us to reuse the weights among different architectures, which largely saves the computational resources.

3.3 Alternative Optimization of Weights and Architecture Hyperparameters

Combining the NES for gradient approximation and weight-sharing in weight training, we proposed to alternatively optimize the weights and the architecture hyperparameters.

In weight training, we start from an initial architecture and add random gaussian variations to the architecture hyperparameter:

$$\min_{\mathcal{W}} \mathcal{L}(\lfloor \mathcal{H} + n \rfloor, \mathcal{W}), n \sim N(0, \sigma). \tag{8}$$

In this way, the weights are trained with respect to various different architectures with a small variance in the architecture hyperparameter. When the architecture hyperparameters varies in a small range, the shared weights in the network can provide reliable loss.

Then we follow the NES for gradient optimization to update the architecture hyperparameters, motivated by the stochastic gradient descent (SGD),

$$\mathcal{H}_{t+1} = \mathcal{H}_t - \alpha \mathcal{G}, \tag{9}$$

where α is the update ratio.

Starting with the updated architecture hyperparameters, the weights can be further trained to adjust new architectures. Alternatively optimize the architecture hyperparameters and the weights until architecture hyperparameters converge. The Algorithm can be found in the Appendix Algorithm 1.

4 Experiments

In this section we verify the effectiveness of our proposed method by comparing our result with state-of-the-art architecture hyperparameter optimization methods.

4.1 Dataset

The proposed method is highly efficient, so it is feasible to carry out all experiments on the ImageNet 2012 classification dataset (Russakovsky et al., 2015).

Table 1: Architecture Hyperparameter Optimization on **MobileNet V1**

Under FLOPs constraints							
		330M		150M		45M	
		Acc	FLOPs	Acc	FLOPs	Acc	FLOPs
Uniform Rescale Baseline	Channel	68.4%	325M	63.7%	149M	50.6%	41M
	Input image	70.6%	343M	65.7%	149M	54.0%	42.5M
	Channel + Input image	70.8%	325M	67.1%	143M	58.8%	45.7M
State-of-the-arts	Netadapt	69.1%	284M	–	–	–	–
	AMC	70.5%	281M	–	–	–	–
	MetaPruning	70.9%	324M	66.4%	149M	57.2%	41.1M
Proposed Method	Channel	71.0%	310M	66.5%	145M	58.6%	47.5M
	Channel + Input image	**71.2%**	309M	**67.6%**	139M	**59.7%**	42.5M

Under GPU latency constraints						
	0.75×		0.5×		0.25×	
	Acc	Latency	Acc	Latency	Acc	Latency
Baseline	68.4%	5.620ms	63.7%	3.998ms	50.6%	2.266ms
Proposed Method	**72.3%**	5.617ms	**70.6%**	3.972ms	**66.8%**	2.231ms

Table 2: Architecture Hyperparameter Optimization on **MobileNet V2**

Under FLOPs constraints							
		200M		150M		45M	
		Acc	FLOPs	Acc	FLOPs	Acc	FLOPs
Uniform Rescale Baseline	Channel	70.0%	203M	67.2%	140M	54.6%	43M
	Input image	70.8%	220M	68.3%	138M	59.1%	42M
	Channel + Input image	70.6%	212M	68.5%	142M	59.3%	47M
State-of-the-arts	AMC	70.8%	220M	–	–	–	–
	MetaPruning	71.2%	217M	68.2%	140M	58.3%	43M
Proposed Method	Channel	70.7%	205M	68.4%	141M	58.1%	46M
	Channel + Input image	**71.4%**	206M	68.8%	139M	60.7%	42M
	Channel + Input image + Depth	70.7%	206M	**69.1%**	145M	**60.9%**	51M

Under GPU latency constraints						
		0.8×		0.65×		0.35×	
		Acc	Latency	Acc	Latency	Acc	Latency
Uniform Rescale Baseline		70.0%	7.36ms	67.2%	5.97ms	54.6%	4.22ms
Proposed Method	Channel + Input image	72.3%	7.34ms	71.2%	5.93ms	67.7%	4.10ms
	Channel + Input image + Depth	**72.4%**	7.17ms	**71.7%**	5.90ms	**68.7%**	3.98ms

Under CPU latency constraints							
	0.75×		0.5×		0.35×		Total Optimization
	Acc	Latency	Acc	Latency	Acc	Latency	Time Cost
ChamNet	**71.9%**	15.0ms	69.0%	10.0ms	64.1%	6.1ms	5760 GPU days
Proposed Method	71.8%	14.8ms	**69.0%**	9.9ms	**66.4%**	6.0ms	**3 × 40 GPU days**

Table 3: Architecture Hyperparameter Optimization on **ResNet50**

		3G		2G		1G	
		Acc	FLOPs	Acc	FLOPs	Acc	FLOPs
	Uniform Baseline	76.0%	3.2G	74.8%	2.3G	72.0%	1.1G
Traditional Purning	AutoPruner (Luo & Wu, 2018)	–	–	74.8%	2.3G	72.0%	1.1G
	ThiNet (Luo et al., 2017)	75.8%	2.9G	74.7%	2.1G	72.1%	1.2G
	CP (He et al., 2017)	–	–	73.3%	2.0G	–	–
	SFP (He et al., 2018a)	75.1%	2.9G	–	–	–	–
AutoML-based	MetaPruning (Liu et al., 2019)	76.2%	3.0G	75.4%	2.0G	73.4%	1.0G
	Our method	**76.2%**	3.0G	**75.6%**	2.0G	**73.4%**	1.0G

ImageNet is a large-scale dataset with 1.2 million training images and 50K validation images of 1000 classes. In our experiments, we split the original training images into sub-evaluation dataset, which contains 50000 images randomly selected from the training images with 50 images in each 1000-class, and sub-training dataset with the rest of images. We optimize the architecture hyperparameters on the sub-evaluation dataset and train the weights on the sub-training dataset. Optimizing the architecture hyperparameters with weight training in the weight-sharing network takes one-fourth the epochs as training the corresponding backbone network from scratch. After the architecture hyperparameters are optimized, we train the corresponding architecture from scratch on the original training dataset and evaluate it on the test dataset. The training details can be found in the Appendix B.

We carry out the optimization on MobileNet v1/v2 (Howard et al., 2017; Sandler et al., 2018) and ResNet (He et al., 2016) backbone. The detailed hyperparameter optimization space can be found in the Appendix C.

4.2 Constraints

We apply the hyperparameter optimization under both the latency and the flops constraint. For the latency constraint, we follow the practice in ChamNet (Dai et al., 2018) and FB-Net (Wu et al., 2018) to build a latency look-up-table for a layer with different hyperparameter choices and obtain the total latency of the network by summing up the latency of all layers in the network. In our experiment we estimate GPU latency on the GTX 1080Ti with batch size of 256. For fair comparison with ChamNet (Dai et al., 2018), we use the look-up-table released by (Dai et al., 2018) as the CPU latency. As the ChamNet has a very sparse look-up-table, we use the gaussian process to predict the missing values as (Dai et al., 2018) did for energy estimation.

4.3 Results Analysis

MobileNet v1 is a network without shortcut and most of convolution layers in MobileNet v1 do not have the same number of output channels, layer dropping will result in a channel number dis-match. With the network structure restriction, we only optimize the channel and spatial dimension for the MobileNet v1. As shown in Table 1, proposed method can learn to allocate the resource properly for achieving higher accuracy. Analyzing the experiment results, there are several interesting observations:

(1) When the optimization goal is only the layer-wise channel numbers, proposed method (10*th* row) achieves much higher accuracy than uniform channel rescaling baselines (4*th* row) as well as the state-of-the-art channel pruning algorithms (Yang et al., 2018; He et al., 2018b; Liu et al., 2019) (7*th*-9*th* rows).

(2) When we further extend the optimization goal to both channel and spatial dimension, the proposed method generates even better results and surpass all the baselines (4*th*-6*th* rows), which suggests the necessity in optimizing different dimensions in a unified way and that our method is capable in handling multiple dimensions.

(3) Moreover, our method can directly optimize with respect to the hardware execution latency, without knowing the implementation details inside the hardware. In optimizing with respect to the GPU latency, we discover that, our algorithm can learn to adapt network with more channels and smaller spatial resolution to take advantage of the highly parallel characteristic of the GPU. In this way we achieved up to 16.2% accuracy enhancement compared to the MobileNet V1 baseline (the last two rows).

MobileNet v2 is a network with shortcut, to handle the channels in the shortcut, we confine the number of output channels to be the same for a sequence of blocks connected with a shortcut. The intermediate channel inside each block can be optimized separately. Our method can easily deal with the channels in the shortcut and produce good results on the MobileNet v2 backbone. Some findings are obtained in the results analysis:

(1) Proposed method (9*th*-10*th* row) consistently achieves much higher accuracy than the corresponding baselines (4*th*-6*th* row), which reveals the effectiveness of the proposed optimization method.

(2) When dealing with the channel, spatial and depth dimension together (11*th* row and 17*th* row), the proposed method can achieve higher accuracy than with optimization goal being solely channel (9*th* row) or channel plus spatial (10*th* row and 16*th* row), which proves the effectiveness of our method in dealing with channel, spatial and depth dimension together and the necessity in jointly optimizing these three dimensions.

(3) Compared with optimization under the flops constraints, latency-oriented optimization can capture the underlying hardware characteristics more easily and customize the network for specific hardware to yield more significant accuracy enhancement. For example, when targeting at the CPU latency, our method learns to generate network with thinner and deeper structure while it chooses wide and shallow structure for a GPU device. This is interpretable, because the GPU is highly paralleled and can execute condensed operations faster than computing fragmented pieces. Thus the accuracy gain is significant when we optimize with all three dimensions (channel + spatial + depth) (17*th* row) than we only optimize two dimensions (channel + spatial) (16*th* row) for a network to be deployed on the GPU device, as shown in Table 2, GPU latency constraint part.

(4) In comparison with the state-of-the-art method, proposed optimization method surpasses the state-of-the-art ChamNet on compact models as 0.35 $\times$ MobileNet v2 and achieved comparable results in larger models with the same look-up table for the CPU latency. Considering ChamNet uses around 5760 GPUs days to train 240 networks for building up the hyperparameter-accuracy curve for each architecture backbone, while our method needs no more time than training a network twice for obtaining one desired optimized network architecture. When targeting at searching one network under specific constraint, proposed method saves the optimization time cost by up to 144 times compared to the state-of-the-art ChamNet.

ResNet is a heavy network with shortcut, we adopt the same channel number constraints as MobileNet v2, the detailed search space can be found in Appendix C. For fair comparison with other methods on the channel pruning, we confine our search space to the channel dimension. The results in Table 3 show that our method outperforms all the traditional channel pruning methods as well as the uniform baseline. Compared to the state-of-the-art AutoML-based algorithm (Liu et al., 2019) specialized in the channel dimension, we can still achieved comparable or higher accuracy. This results show that although our method is proposed for jointly optimizing three dimensions of the architecture hyperparameters, it still achieve no inferior results than other specialized-purpose algorithms when only targeting at one dimension.

5 CONCLUSION

In this paper, we proposed a unified gradient-based method for architecture hyperparameter optimization which contains following advantages: (1) it optimizes the architecture hyperparameters in channel, spatial and depth dimensions in a unified way; (2) it can be easily incorporated into the stochastic gradient descent framework; (3) constraints such as hardware latency can be effortlessly handled; (4) it achieves state-of-the-art results and is highly efficient.

REFERENCES

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pp. 3981–3989, 2016.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pp. 549–558, 2018.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pp. 2546–2554, 2011.

Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.

Lars Buesing, Theophane Weber, and Shakir Mohamed. Stochastic gradient estimation with finite differences. 2016.

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. *arXiv preprint arXiv:1812.08934*, 2018.

Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Xiaohan Ding, Guiguang Ding, Yuchen Guo, and Jungong Han. Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4943–4953, 2019a.

Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan. Approximated oracle filter pruning for destructive cnn width optimization. *arXiv preprint arXiv:1905.04748*, 2019b.

Edward Farhi, Jeffrey Goldstone, Sam Gutmann, Joshua Lapan, Andrew Lundgren, and Daniel Preda. A quantum adiabatic evolution algorithm applied to random instances of an np-complete problem. *Science*, 292(5516):472–475, 2001.

Michael C Fu. Gradient estimation. *Handbooks in operations research and management science*, 13:575–616, 2006.

Michael C Fu. Stochastic gradient estimation. In *Handbook of simulation optimization*, pp. 105–147. Springer, 2015.

Tobias Glasmachers, Tom Schaul, Sun Yi, Daan Wierstra, and Jürgen Schmidhuber. Exponential natural evolution strategies. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 393–400. ACM, 2010.

Paul Glasserman and Yu-Chi Ho. *Gradient estimation via perturbation analysis*, volume 116. Springer Science & Business Media, 1991.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*, 2019.

Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018a.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1389–1397, 2017.

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 *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 784–800, 2018b.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 304–320, 2018.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pp. 507–523. Springer, 2011.

Jan Koutnik, Faustino Gomez, and Jürgen Schmidhuber. Evolving neural networks in compressed weight space. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 619–626. ACM, 2010.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Yingzhen Li and Richard E Turner. Gradient estimators for implicit models. *arXiv preprint arXiv:1705.07107*, 2017.

Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018a.

Junhong Liu and Jouni Lampinen. A fuzzy adaptive differential evolution algorithm. *Soft Computing*, 9(6):448–462, 2005.

Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Tim Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. *arXiv preprint arXiv:1903.10258*, 2019.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018b.

Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *arXiv preprint arXiv:1805.08941*, 2018.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pp. 5058–5066, 2017.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pp. 2113–2122, 2015.

Rammohan Mallipeddi, Ponnuthurai N Suganthan, Quan-Ke Pan, and Mehmet Fatih Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied soft computing*, 11(2):1679–1696, 2011.

Zelda Mariet and Suvrit Sra. Diversity networks. *Proceedings of ICLR*, 2016.

A Kai Qin and Ponnuthurai N Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *2005 IEEE congress on evolutionary computation*, volume 2, pp. 1785–1791. IEEE, 2005.

A Kai Qin, Vicky Ling Huang, and Ponnuthurai N Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, 13(2):398–417, 2008.

Ingo Rechenberg. *Evolutionsstrategie'94*. frommann-holzboog, 1994.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Tom Schaul, Tobias Glasmachers, and Jürgen Schmidhuber. High dimensions and heavy tails for natural evolution strategies. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 845–852. ACM, 2011.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pp. 3528–3536, 2015.

Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

Jiaxin Shi, Shengyang Sun, and Jun Zhu. A spectral approach to gradient estimation for implicit distributions. *arXiv preprint arXiv:1806.02925*, 2018.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012.

Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. *arXiv preprint arXiv:1904.02877*, 2019.

Yi Sun, Daan Wierstra, Tom Schaul, and Juergen Schmidhuber. Efficient natural evolution strategies. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 539–546. ACM, 2009.

Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.

Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 3381–3387. IEEE, 2008.

Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2014.

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. *arXiv preprint arXiv:1812.03443*, 2018.

Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. *arXiv preprint arXiv:1904.01569*, 2019.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 285–300, 2018.

Jiahui Yu and Thomas Huang. Universally slimmable networks and improved training techniques. *arXiv preprint arXiv:1903.05134*, 2019.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

A Optimization Algorithm

The proposed architecture hyperparameter optimization algorithm is as follows:

Algorithm 1 Unified Gradient-based Architecture Hyperparameter Optimization Algorithm

Notation: Loss: $\mathcal{L}$.
Parameters: Weight: $\mathcal{W}$, Architecture hyperparameter: $\mathcal{H}$.
Training Parameters: Number of total iterations: $\mathcal{K}$, Number of weight training iterations: $\mathcal{N}$, Number of architecture hyperparameter training iterations: $\mathcal{M}$
Input: Backbone architecture: $\mathcal{A}$
Output: Optimized architecture hyperparameter $\mathcal{H}^*$.

$\mathcal{A}_0 = \mathcal{A}(\mathcal{H}_0)$ *# Initial architecture defined with hyperparameter.*
for $t = 0 : \mathcal{K}$ **do**
  # Train the weights for different architecture hyperparameters
  for $i = 0 : \mathcal{N}$ **do**
    $\min_{\mathcal{W}} \mathcal{L}(\mathcal{W}, int(\mathcal{H} + n)), n \sim N(0, \sigma)$
  end for
  # Optimizing the architecture hyperparameter with Natural Evolutionary Strategy
  for $j = 0 : \mathcal{M}$ **do**
    $\Delta\mathcal{L}_j = \mathcal{L}(\mathcal{W}, int(\mathcal{H}_t + n_j)) - \mathcal{L}(\mathcal{W}, \mathcal{H}_t), \;\; n_j \sim N(0, \sigma)$
    $\mathcal{H}_{t+1} \leftarrow \mathcal{H}_t$ - $\alpha \frac{1}{\sigma \mathcal{M}} \sum_{j=0}^{\mathcal{M}} \Delta\mathcal{L}_j n_j$
  end for
end for

B Training Details

For alternative optimizing the architecture hyperparameters and weights, in each inner loop, we train the weights for 2000 iterations for MobileNet v1/v2 and 1000 iterations for ResNet with a batchsize of 256. Then we update the architecture hyperparameters for 20 iterations with the gradient approximated by the natural evolutionary strategy (NES). In NES, the total number $\mathcal{M}$ of the attempted architecture variation is 100 for each iteration, the deviation σ of the guassian kernel for the variance adding to the architecture hyperparameters is initialized with 1.25 and linearly decay to 0. The learning rate α for architecture hyperparameter update is initialized with 5 and linearly decay to 0. We alternative train the weights and update the architecture hyperparameters with an outer loop of 75 iterations. The weight training takes up about 64 epochs for MobileNet v1/v2 and 32 epochs for ResNet50.

C Hyperparameter Optimization Space

The hyperparameter optimization space for MobileNet V1/V2 and ResNet is summarized in Table 4. For the depth dimension, we drop the last repeated block in the network to decrease the depth.

Table 4: Hyperparameter (HP) optimization space.

		Number of HPs	Range	Step
MobileNet V1	Channel	13	$0.1\times C_{base} \rightarrow 1.6\times C_{base}$	$0.03\times C_{base}$
	Spatial	1	96 pixels $\rightarrow$ 224 pixels	1
MobileNet V2	Channel in shortcut	8	$0.2\times C_{base} \rightarrow 1.5\times C_{base}$	$0.05\times C_{base}$
	Middle channel in block	17	$0.2\times C_{base} \rightarrow 1.5\times C_{base}$	$0.05\times C_{base}$
	Spatial	1	96 pixels $\rightarrow$ 224 pixels	1
	Depth	1	9 blocks $\rightarrow$ 19 blocks	1
ResNet	Channel in shortcut	4	$0.1\times C_{base} \rightarrow 1.6\times C_{base}$	$0.03\times C_{base}$
	Middle channel in block	16	$0.1\times C_{base} \rightarrow 1.6\times C_{base}$	$0.03\times C_{base}$

*C_{base}: the base channel in each layer of the 1x backbone network.