Neural Epitome Search for Architecture-Agnostic Network Compression

Anonymous authors
Paper under double-blind review

Abstract

Traditional compression methods including network pruning, quantization, low rank factorization and knowledge distillation all assume that network architectures and parameters should be hardwired. In this work, we propose a new perspective on network compression, i.e., network parameters can be disentangled from the architectures. From this viewpoint, we present the Neural Epitome Search (NES), a new neural network compression approach that learns to find compact yet expressive epitomes for weight parameters of a specified network architecture end-to-end. The complete network to compress can be generated from the learned epitome via a novel transformation method that adaptively transforms the epitomes to match shapes of the given architecture. Compared with existing compression methods, NES allows the weight tensors to be independent of the architecture design and hence can achieve a good trade-off between model compression rate and performance given a specific model size constraint. Experiments demonstrate that, on ImageNet, when taking MobileNetV2 as backbone, our approach improves the full-model baseline by 1.47% in top-1 accuracy with 25% MAdd reduction and AutoML for Model Compression (AMC) by 2.5% with nearly the same compression ratio. Moreover, taking EfficientNet-B0 as baseline, our NES yields an improvement of 1.2% but are with 10% less MAdd. In particular, our method achieves a new state-of-the-art results of 77.5% under mobile settings (<350M MAdd). Code will be made publicly available.

1 Introduction

Despite the remarkable performance achieved in many applications, powerful deep convolutional neural networks (CNNs) typically suffer from high complexity (Han et al., 2015a). The large model size and computation cost hinders their deployment on resource limited devices, such as mobile phones. Very recently, huge efforts have been made to compress powerful CNNs. Existing compression techniques can be generally categorized into four categories: network pruning (Han et al., 2015a; Collins & Kohli, 2014; Han et al., 2015b), low rank factorization (Jaderberg et al., 2014), quantization (Jacob et al., 2018; Hubara et al., 2017; Rastegari et al., 2016), and knowledge distillation (Hinton et al., 2015; Papernot et al., 2016). Network pruning targets on removing unimportant connections or weights to reduce the number of parameters and multiply-adds (MAdd). Low rank factorization decomposes an existing layer into lower-rank and smaller layers to reduce the computation cost. Weights quantization aims to use less number of bits to store the weights and activation maps. Knowledge distillation uses a well trained teacher network to train a lightweight student network. All of those compression methods assume that the model parameters (weight tensors) must have one-to-one correspondence to the architectures. As a result, they suffer from performance drop since changing architectures will inevitably lead to loss of informative parameters.

In this paper, we consider the network compression problem from a new perspective where the shape of the weight tensors and the architecture are designed independently. The key insight is that the network parameters can be disentangled from the architecture and can be compactly represented by a small-sized parameter set (called epitome), inspired by success of epitome methods in image/video modeling and data sparse coding (Jojic et al., 2003; Cheung et al., 2008; Aharon & Elad, 2008). As shown in Figure 1, unlike conventional convolutional layers that use the architecture tied weight tensors to convolve with the input feature map, our proposed neural epitome search (NES) approach first learns a compact yet expressive epitome along with an adaptive transformation function to

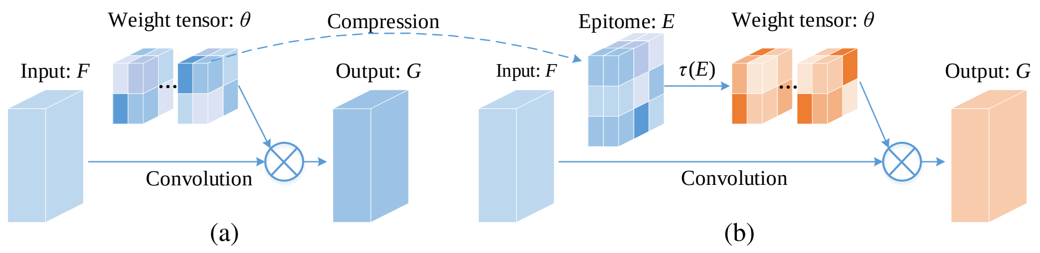

Figure 1: (a) illustrates the conventional convolution process; (b) shows the convolution with NES method. Epitome E has a different shape as defined by the architecture. A transformation is learned automatically to transform the E to a shape that match the architecture defined shape. By designing a smaller E, significant compression can be achieved with less performance drop. In certain cases, the performance can be increased with less computation as shown in Table 3.

expand the epitomes. The transformation function is able to generate a variety of parameters from epitomes via a novel learnable transformation function, which also guarantees the representation capacity of the resulting weight tensors to be large. Our transformation function is differentiable and hence enables the NES approach to search for optimal epitome end-to-end, achieving a good trade-off between required model size and performance. In addition, we propose a novel routing map to record the index mapping used for the transformation between the epitome and the convolution kernel. During inference, this routing map enables the model to reuse computations when the expanded weight tensors are formed based on the same set of elements in the epitomes and therefore effectively reduces the computation cost.

Benefiting from the learned epitomic network parameters and transformation method, compared to existing compression approaches, NES has less performance drop. To the best of our knowledge, this is the first work that automatically learns compact epitomes of network parameters and the corresponding transformation function for network compression. To sum up, our work offers the following attractive properties:

- Our method is flexible. It allows the weight tensors to be independent of the architecture design. We can easily control the model size by defining the size of the epitomes given a specified network architecture. This is especially beneficial in the context of edge devices.
- Our method is effective. The learning-based transformation method empowers the epitomes with highly expressive capability and hence incurs less performance drop even with large compression ratios.
- Our method is easy to use. It can be encapsulated as a drop in replacement to the current convolutional operator. There is no dependence on specialized platforms/frameworks for NES to conduct compression.

To demonstrate the efficacy of the proposed approach, We conduct extensive experiments on CIFAR-10 (Krizhevsky & Hinton, 2009) and ImageNet (Deng et al., 2009). On CIFAR-10 dataset, our method outperforms the baseline model by 1.3%. On ImageNet, our method outperforms MobileNetV2 full model by 1.47% in top-1 accuracy with 25% MAdd reduction, and MobileNetV2-0.35 baseline by 3.78%. Regarding MobileNetV2-0.7 backbone, our method improves AMC (He et al., 2018) by 2.47%. Additionally, when taking EfficientNet-b0 (Tan & Le, 2019) as baseline, we have an improvement of 1.2% top-1 accuracy with 10% MAdd reduction.

2 Related work

Traditional model compression methods include network pruning (Collins & Kohli, 2014; Han et al., 2015b), low rank factorization (Jaderberg et al., 2014), quantization (Jacob et al., 2018; Hubara et al., 2017; Rastegari et al., 2016) and knowledge distillation (Hinton et al., 2015; Papernot et al., 2016). For all of those methods, as mentioned in Section 1 extensive expert knowledge and manual efforts are needed and the process might need to be done iteratively and hence is time consuming.

Recently, AutoML based methods have been proposed to reduce the experts efforts for model compression (He et al., 2018; Zoph et al., 2018; Noy et al., 2019; Li et al., 2019) and efficient convolution architecture design (Liu et al., 2018; Wu et al., 2018; Tan et al., 2018). As proposed in

AutoML for model compression (AMC (He et al., 2018)), reinforcement learning can be used as an agent to remove redundant layers by adding resource constraints into the rewards function which however is highly time consuming. Later, gradient based search method such as DARTS (Liu et al., 2018) is developed for higher search efficiency over basic building blocks. There are also methods that use AutoML based method to search for efficient architectures directly (Wu et al., 2018; He et al., 2018). All of those methods are searching for optimized network architecture with an implicit assumption that the weights and the model architecture have one-to-one correspondence. Different from all of the above mentioned methods, our method provides a new search space by separating the model weights from the architecture. The model size can thus be controlled precisely by nature.

Our method is also related to the group-theory based network transformation. Based on the group theory proposed in (Cohen & Welling, 2016), recent methods try to design a compact filter to reduce the convolution layer computation cost such as WSNet (Jin et al., 2017) and CReLU (Shang et al., 2016). WSNet tries to reduces the model parameters and computations by allowing overlapping between adjacent filters of 1D convolution kernel. This can be seen as a simplified version of our method as the overlapping can be regarded as a fixed rule transformation. CReLu tries to learn diversified features by concatenating ReLU output of original and negated inputs. However, as the rule is fixed, the design of those schemes are application specific and time consuming. Besides, the performance typically suffers since the scheme is not optimized during the training. In contrast, our method requires negligible human efforts and the transformation rule is learned end-to-end.

3 METHOD

3.1 OVERVIEW

A convolutional layer is composed of a set of learnable weights (or kernels) that are used for feature transformation. The observation of this paper is that the learnable weights in CNNs can be disentangled from the architecture. Inspired by this fact, we provide a new perspective on the network compression problem, i.e., finding a compact yet expressive parameter set, called *epitome*, along with a proper transformation function to fully represent the whole network, as illustrated in Figure 1.

Formally, consider a convolutional network with a fixed architecture consisting of a stack of convolutional layers, each of which is associated with a weight tensor θ_i (i is layer index). Further, let $\mathcal{L}(X, Y; \theta)$ be the loss function used to train the network, where X and Y are the input data and label respectively and $\theta = \{\theta_i\}$ denotes the parameter set in the network. Our goal is to learn epitomes $E = \{E_i\}$ which have smaller sizes than θ and the transformation function $\tau = \{\tau_i\}$ to perform mapping from $\tau(E)$ to θ. In this way, network compression is achieved. The objective function of neural epitome search (NES) for a given architecture is to learn optimal epitomes E^* and transformation functions τ^*:

$$\{E^*, \tau^*\} = \underset{E,\tau}{\arg\min}\, \mathcal{L}(X, Y; \tau(E)), \qquad s.t. \quad |E| < |\theta|, \tag{1}$$

where $|\cdot|$ calculates the number of all elements.

The above NES approach provides a flexible way to achieve network compression since the epitomes can be defined to be of any size. By learning a proper transformation function, the epitomes of predefined sizes can be adaptively transformed to match and instantiate the specified network architecture. In the following sections, we will elaborate on how to learn the transformation functions $\tau(\cdot)$, the epitomes E and how to compress models via NES in an end-to-end manner.

3.2 DIFFERENTIABLE SEARCH FOR EPITOME TRANSFORMATION

To implement the proposed NES, one challenge is how to make the transformation function, τ, which maps from the epitome to convolutional kernels differentiable. A simple choice for τ is a linear transformation function that applies fixed and predefined transformation rules as done in (Shang et al., 2016; Jin et al., 2017). However, those rules are too rigid to adaptively express different networks and would hurt the performance after compression. Here, we carefully examine the indexing operations and propose a new operation that can make the transformation differentiable with strong flexibility.

A conventional 2D convolutional layer with an input feature map $F \in \mathbb{R}^{W \times H \times C_{in}}$, produces an output feature map $G \in \mathbb{R}^{W \times H \times C_{out}}$, where (W, H, C_{in}, C_{out}) denote the width, height, input

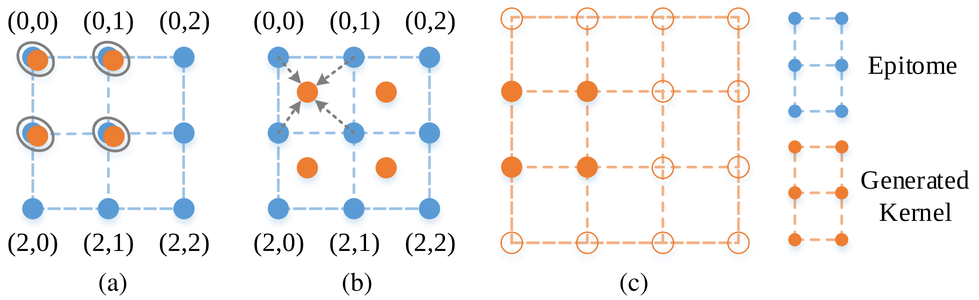

Figure 2: (a) A straightforward way to learn integer starting indices to generate kernels which is non-differentiable; (b): Our learnable interpolation method in which the starting indices can be fractional as demonstrated in Eqn. 3; (c) Weight kernel used in full-sized network that needs to be generated. Blue/orange nodes represent epitome/generated parameters respectively.

channels and output channels, respectively. Let f_{t_w,t_h} be a patch of the input feature map, which spatially spans from t_w to $t_w + w - 1$ and t_h to $t_h + h - 1$, and k_c be the c-th filter of the convolution kernel $\mathcal{K}$ parameterized by weights $\theta \in \mathbb{R}^{w \times h \times C_{in} \times C_{out}}$, where w and h are the spatial size of the convolution kernel. Here, we misuse the notion of θ by ignoring the layer index subscript i to simplify the equation as each layer is processed independently. The convolution can be formulated as

$$G_{t_w,t_h,c} = f_{t_w,t_h} * k_c = \sum_{i}^{w} \sum_{j}^{h} \sum_{m}^{C_{in}} F_{t_w+i,t_h+j,m} \theta_{i,j,m,c}. \tag{2}$$

To achieve model compression, the epitome with shape $(W', H', C'_{in}, C'_{out})$ is constrained to have less elements than the full-sized kernel weights as shown in Eqn. 1. A straightforward way to fill in the full weight parameters from the epitome would be learning an index mapping from the epitome to the convolutional kernel as shown in Figure 2(a) first and then concatenating the selected elements to recover the convolutional kernel. However, the above operation of indexing the epitome is non-differentiable. Moreover, the outputs of parameterized transformation functions (e.g. neural networks) are typically fractional and hence cannot be directly used as mapping indices of epitomes.

To handle the above two obstacles, we present an efficient transformation function, including three parts: (1) a parameterized transformation learner η used to learn a set of starting indices for patches in epitome (i.e. all elements in the same epitome patch share identical starting indices); (2) a routing map $\mathcal{M}$ that records the location mapping from the epitome to the convolution kernel; (3) an interpolation-based generator (Eqn. 3). In what follows, for easy understanding, we take the spatial dimensions as an example to demonstrate the transformation process.

For some location (i, j) in the convolution kernel, the transformation learner η produces a pair of starting indices (p, q) along the width and height dimensions. The location mapping from the learned indices (p, q) to (i, j) is recorded in $\mathcal{M}$. We update $\mathcal{M}$ in a moving average way so that η can be removed during inference. The element value of the kernel at location (i, j) can be generated by:

$$\theta_{(i,j)} = \tau(E) = \sum_{n_w=0}^{W'} \sum_{n_h=0}^{H'} \mathcal{G}(n_w, p)\mathcal{G}(n_h, q)E_{(n_w,n_h)}, \tag{3}$$

where $\mathcal{G}(a, b) = \max(0, 1 - |a - b|)$ and n_w and n_h enumerate over all integral spatial locations within E. It is worth mentioning that in Eqn. 3 the generated indices can be real values. Figure 2(b) shows a brief illustration of the above process, in which the size of an epitome patch is 2×2 (the orange nodes). The above process can be extended to 4D space intuitively when the epitome is of four dimensions.

3.3 Learning to Search Epitomes End-to-end

Benefiting from the differentiable Eqn. 3, the elements in E and the transformation learner η can be updated together with the convolutional layers through back propagation in an end-to-end manner. For the epitome E, as its transformed parameter $E_{(p,q)}$ can be used in multiple positions, the gradients of the epitomes are the summation of all the positions where the weight parameters are used.

Here, for clarity, we use $\{\tau^{-1}(p,q)\}$ to denote the set of the indices in the convolution kernel that are mapped from the same position (p,q) in E. The gradients of $E_{(p,q)}$ can, thus, be updated as:

$$\nabla_{E_{(p,q)}}\mathcal{L} = \sum_{z\in\{\tau^{-1}(p,q)\}} \alpha_z \nabla_{K_z}\mathcal{L}, \tag{4}$$

where K_z is the kernel parameters that are transformed from $E_{(p,q)}$, and α_z are the fractions that are assigned to $E_{(p,q)}$ during the transformation.

The transformation learner η consists of two convolutional layer, followed by a sigmoid function. To index the whole epitome, the output value along each dimension is scaled by the size of the corresponding dimension. As it takes the input feature map of the convolutional layer as input, the parameters in η can be simply updated according to the chain rule.

3.4 Compression Efficiency

Parameter reduction. By using the routing map which records location mappings from epitome to convolution kernel, the total number of parameters during inference is decided by the size of the epitome. Assume that the epitome E is a four dimensional tensor. Recall that E is with shape $(W', H', C'_{in}, C'_{out})$. The size of an epitome patch is denoted as $w \times h \times \beta_1 \times \beta_2$[1] ($\beta_1 \leq C'_{in}$ and $\beta_2 \leq C'_{out}$). Note that we can enlarge the size of epitome patch to reduce memory costs. See details in the Appendix D. When performing the transformation along all four dimensions, NES achieves the following compression ratio r:

$$r = \frac{w \times h \times C_{in} \times C_{out}}{W' \times H' \times C'_{in} \times C'_{out} + 3 \times R_{cin} + R_{cout}}, \tag{5}$$

where $R_{cout} = \lceil C_{out}/\beta_2 \rceil$ is the number of interpolations applied to the output channel dimension, and $R_{cin} = \lceil C_{in}/\beta_1 \rceil$ is the compression ratio along the input channel dimension between the convolution kernel and the epitome. From Eqn. 5, it can be obviously seen that the compression ratio is nearly proportional to $\frac{C_{out}\times C_{in}\times w\times h}{C'_{out}\times C'_{in}\times W'\times H'}$, the ratio between the size of the epitome and the generated weight tensor. Detailed proof can be found in Appendix D. The above analysis demonstrates that NES provides a *precise control of the model size* via the proposed transformation function.

Computation reduction. As the weight tensor θ is generated from the epitome E, the computation in convolution can be reused when different elements in θ are from the same portion of elements in E. Concretely, given the routing map $\mathcal{M}$ for the transformation, we first calculate the convolution results between the epitome and the input feature map once and then save the results as a product map P. However, directly keeping the intermediate results will take a huge resource of the computation memory. We introduce two techniques to reduce the size of P. Along the channel dimension, we apply channel wrap (see details in Appendix D). Along the filter dimension, we set a small value of β_2. During inference, given P, the multiplication in convolution can be reused in a lookup table manner with $O(1)$ complexity:

$$G_{t_w,t_h,c} = \sum_i^W \sum_j^H \sum_m^{C_{in}} F_{t_w+i,t_h+j,m}\mathcal{K}_{\mathcal{M}(i,j,m,c)} = \sum_i^W \sum_j^H \sum_m^{C_{in}} P_{\mathcal{M}(i,j,m,c)}. \tag{6}$$

The additions in Eqn. 6 can also be reused via an integral map I (Crow, 1984) as done in (Viola et al., 2001). With the product map and the integral map, the MAdd reduction ratio can be written as:

$$\text{MAdd Reduction Ratio} = \frac{C_{out}HW(2C_{in}wh - 1)}{C'_{out}HW(W_cH_c + 2C'_{in}W_cH_c - 1) + 2R_{cin}\beta_1 + 2\beta_2 R_{cout}}. \tag{7}$$

See more details and analysis in Appendix D.

Discussion. We make a few remarks on the advantages of our proposed method as follows. The proposed NES method disentangles the weight tensors from the architecture by using a learnable transformation function. This provides a new research direction for model compression by bringing in better design flexibility against the traditional compression methods on both sides of software and

[1] We set β_1 to C'_{in} and β_2 to $C'_{out}/2$ in this paper.

Table 1: Comparison with WSNet on ESC-50 dataset. We use the same configuration but change the sampling stride to be learnable. 'S' denotes the stride and 'C' denotes the repetition times along the input channel dimension. We use the 'S' in WSNet as initial values and learns the offsets.

Method	Conv1		Conv2		Conv3		Conv4		Conv5		Conv6		Conv7		Conv8		Acc. (%)
Config.	S	C	S	C	S	C	S	C	S	C	S	C	S	C	S	C	
baseline	1	1	1	1	1	1	1	1	S	1	1	1	1	1	1	1	66.0
WSNet	8	1	4	1	2	2	1	2	S	4	1	8	1	8	1	8	66.5
Ours	8	1	4	1	2	2	1	2	S	4	1	8	1	8	1	8	**73.0**

Table 2: Results of ImageNet classification. Our method uses vanilla MobileVetV2 as backbone. For fair comparison, we evaluate multiple width multiplier values of $0.75, 0.5, 0.35$ and 0.18 and only apply it on the filter dimension of the first 1×1 convolution. We apply the proposed method on all the invert residual blocks equally to disentangle the architecture affects on the performance. MAdd are calculated based on all convolution blocks with an assumption that the batch normalization layers are merged. '$*$' denotes our own implementation.

Methods	MAdd(M)	Parameters	Param Compression Rate	Top-1 Accuracy(%)
MobilenetV2-1.0	301	3.4M	1.00×	71.8
MobilenetV2-0.75*	217	2.94M	1.17×	69.14
MobilenetV2-0.5*	153	2.52M	1.36×	67.22
MobilenetV2-0.35*	115	2.26M	1.54×	65.18
MobilenetV2-0.18*	71	1.98M	1.80×	60.70
Our method-0.75	220	2.94M	1.17×	71.54
Our method-0.5	157	2.52M	1.36×	69.42
Our method-0.35	120	2.26M	1.54×	67.01
Our method-0.18	79	1.95M	1.80×	64.48

hardware. On the software side, NES needs no special acceleration algorithms. All the operations employed by NES are compatible with popular neural network libraries and can be encapsulated as a drop in operator. On the hardware side, the memory allocation of NES is more flexible. This is especially helpful for hardware platform design where the off chip memory access is the main power consumption as demonstrated in (Han et al., 2016). NES provides a way to balance the computation/memory-access ratio in hardware: a smaller epitome with a complex transformation function results in a computation intense model while a large epitome with simple transformation function results in a memory intensive model. Such ratio is an important hardware optimization criteria which however is not covered by most previous compression methods.

4 Experiments

We first evaluate the efficacy of our method in 1D convolutional model compression on the sound dataset ESC-50 (Piczak, 2015) for the comparison with WSNet. We then test our method with MobileNetV2 and EfficientNet as the backbone on 2D convolutions on ImageNet dataset (Deng et al., 2009) and CIFAR-10 dataset (Krizhevsky & Hinton, 2009). Detailed experiments settings can be found in Appendix A. computation cost, and classification performance. For all experiments, we do not use additional training tricks including the squeeze-and-excitation module (Hu et al., 2018) and the Swish activation function (Ramachandran et al., 2017) which can further improve the results. The calculation of MAdd is performed for all convolution blocks. We evaluate our methods in terms of three criteria: model size, multiply-adds(MAdd) and the classification performance.

4.1 1D CNN compression

For 1D convolution compression, we compare with WSNet. Similar to WSNet (Jin et al., 2017), we use the same 8-layer CNN model for fair comparisons. The compression ratio in WSNet is decided by the stride (S) and the repetition times along the channel dimension (C), as shown in Table 1. From Table 1, one can see that with the same compression ratio, our method outperforms WSNet by 6.5% in classification accuracy. This is because our method is able to learn proper weights and learn a

Table 3: Comparison of our method with other state-of-the-art models on ImageNet where our method shows superior performance over all other methods. MAdd are calculated based on all convolution blocks with an assumption that the batch normalization layers are merged. Suffix '-A' means we use larger compression ratio for front layers. Our method does not modify the backbone model architecture and applies a uniform compression ratio, unless specified with suffix '-A'. All experiments are using MobileNetV2 as backbone unless labeled with EfficientNet as suffix.

GROUP	Methods	MAdd (M)	Params	Top-1 Acc. (%)
60M MAdd	MobilenetV2-0.35 (Sandler et al., 2018)	59	1.7M	60.3
	S-MobilenetV2-0.35 (Yu et al., 2018)	59	3.6M	59.7
	US-MobilenetV2-0.35 (Yu & Huang, 2019b)	59	3.6M	62.3
	MnasNet-A1 (0.35x) (Tan et al., 2018)	63	1.7M	62.4
	Our method-0.18	79	2.0M	**64.48**
100M MAdd	MobilenetV2-0.5 (Sandler et al., 2018)	97	2.0 M	65.4
	S-MobilenetV2-0.5 (Yu et al., 2018)	97	3.6M	64.4
	US-MobilenetV2-0.5 (Yu & Huang, 2019b)	97	3.6M	65.1
	Our method-0.35	120	2.2M	**67.01**
200M+ MAdd	MobilenetV2-0.75 (Sandler et al., 2018)	209	2.6 M	69.8
	S-MobilenetV2-0.75 (Yu & Huang, 2019a)	209	3.6M	68.9
	US-MobilenetV2-0.75 (Yu & Huang, 2019b)	209	3.6M	69.6
	FBNet-A (Wu et al., 2018)	246	4.3M	73
	AUTO-S-MobilenetV2-0.75 (Yu & Huang, 2019a)	207	4.1M	73
	Our method-0.5	**157**	**2.5M**	**69.42**
	Our method-0.75	**220**	**2.9M**	**71.54**
	Our method-0.75-A	**225**	**3.7M**	**73.27**
	Our method-0.5 (EfficientNet-b0)	**240**	**3.92M**	**75.55**
	Our method-0.5 (EfficientNet-b1)	**350**	**5.46M**	**77.5**

transformation rules that are adaptive to the dataset of interest and thus overcome the limitation of WSNet where the sampling stride is fixed. More results can be found in Appendix B.

4.2 2D CNN COMPRESSION

Implementation details. We use both MobilenetV2 (Sandler et al., 2018) and EfficientNet (Tan & Le, 2019) as our backbones to evaluate our approach on 2D convolutions. Both models are the most representative mobile networks very recently and have achieved great performance on ImageNet and CIFAR-10 datasets with much fewer parameters and MAdd than ResNet (He et al., 2016) and VGGNet (Simonyan & Zisserman, 2014). For fair comparisons, we follow the experiment settings as in the original papers.

Results on ImageNet. We first conduct experiments on the ImageNet dataset to investigate the effectiveness of our method. We use the same width multiplier as in (Sandler et al., 2018) as our baseline. We choose four common width multiplier values, *i.e.*, $0.75, 0.5, 0.35$ and 0.18 (Sandler et al., 2018; Zhang et al., 2018; Yu et al., 2018) for fair comparison with other compression approaches.

The performance of our method and the baseline is summarized in Table 2. For all the width multiplier values, our method outperforms the baseline by a significant margin. It can be observed that under higher compression ratio the performance gain is also larger. Moreover, the performance of MobileNetV2 drops significantly when the compression ratio is larger than $3\times$. However, our NES method increases the performance by 3.78% at a large compression ratio. This is because when the compression ratio is high, each layer in the baseline model does not have enough capacity to learn good representations. Our method is able to generate more expressive weights from the epitome with a learned transformation function.

We also compare our method with the state-of-the-art compression methods in Table 3. Since an optimized architecture tends to allocate more channels to upper layers (He et al., 2018; Yu & Huang, 2019a), we also run experiments with larger size of the epitome for upper layers. The results is denoted with suffix '-A' in Table 3. Comparison with more models are shown in Figure 3. As shown, NES outperforms the current SOTA mobile model (less than 400M MAdd model) EfficientNet-b0 by 1.2% with 40M less MAdd. Obviously, our method performs even better than some NAS-based

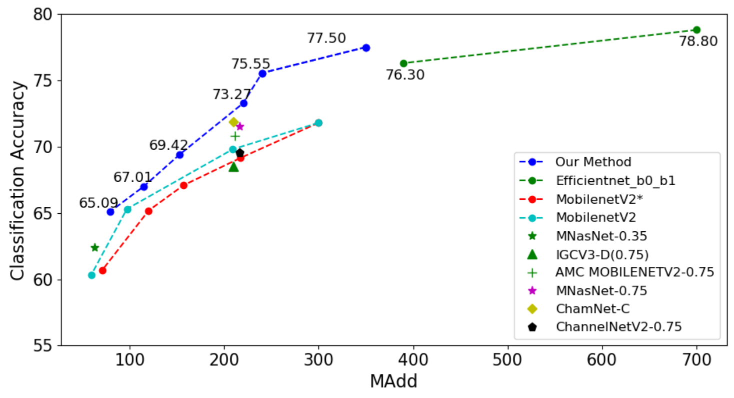

Figure 3: ImageNet classification accuracy of our method, EfficientNet, MobileNetV2 baselines and other NAS based methods including AMC (He et al., 2018), IGCV3 (Sun et al., 2018), MNasNet (Tan et al., 2018), ChamNet (Dai et al., 2018) and ChannelNet (Gao et al., 2018). Our method outperforms all the methods within the same level of MAdd. Here, MobileNetV2* is our implementation of baseline models and MobileNetV2 is the original model with width multiplier of 0.35,0,5,0,75 and 1. The backbone model for our results are EfficientNet_b1 and b0 (Tan & Le, 2019) with multiplier 0.5 and MobileNetV2 with multiplier of 0.75, 0.5, 0.35 and 0.2, respectively (from top to bottom). Note that we do not use additional training tricks including the squeeze-and-excitation module (Hu et al., 2018) and the Swish activation function (Ramachandran et al., 2017).

Table 4: Comparison with other state-of-the-art models in classification accuracy on CIFAR-10. Our method outperforms the recently proposed AUTO-SLIM (Yu & Huang, 2019a) which is a compression method using AutoML method.

Methods	Parameters	MAdd (M)	Top-1 Accuracy(%)
MobilenetV2-1	2.2M	93.52	94.06*
MobilenetV2-0.18*	0.4 M	21.59	91.70
Auto-Slim	0.7M	59	93.00
Auto-Slim	0.3M	28	92.00
Our method	0.39M	26.8	**93.06**

methods.Although our method does not modify the model architecture, the transformation from the epitome to the convolution kernel optimizes the parameter allocation and enriches the model capacity through the learned weight combination and sharing.

Results on CIFAR-10. We also conduct experiments on CIFAR-10 dataset to verify the efficiency of our method as shown in Table 4. Our method achieves 3.5$\times$ MAdd reduction and 5.64$\times$ model size reduction with only 1% accuracy drop, outperforming NAS-based AUTO-SLIM (Yu & Huang, 2019a). More experiments and implementation details are shown in the supplementary material.

Discussion. From the above results, one can observe significant improvements of our method over competitive baselines, even the latest architecture search based methods. The improvement of our method mainly comes from alleviating the performance degradation due to insufficient model size by learning richer and more reasonable combination of weights parameters and allocating suitable weight parameters sharing among different filters. The learned transformation from the epitome to the convolution kernel increases the weight representation capability with less increase on the memory footprint and the computation cost. This distinguishes our method from previous compression methods and improves the model performance significantly under the same compression ratio.

5 Conclusion

We present a novel neural epitome search method which can reuse the parameters efficiently to reduce the model size and MAdd with minimum classification accuracy drop or even increased accuracy in certain cases. Motivated by the observation that the parameters can be disentangled form the architecture, we propose a novel method to learn the transformation rule between the filters to make the transformation adaptive to the dataset of interest. We demonstrate the effectiveness of the method on CIFAR-10 and ImageNet dataset with extensive experimental results.

References

Michal Aharon and Michael Elad. Sparse and redundant modeling of image content using an image-signature-dictionary. *SIAM Journal on Imaging Sciences*, 1(3):228–247, 2008.

Vincent Cheung, Brendan J Frey, and Nebojsa Jojic. Video epitomes. *International Journal of Computer Vision*, 76(2):141–152, 2008.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pp. 2990–2999, 2016.

Maxwell D Collins and Pushmeet Kohli. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*, 2014.

Franklin C Crow. Summed-area tables for texture mapping. In *ACM SIGGRAPH Computer Graphics*, volume 18, pp. 207–212. ACM, 1984.

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.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Channelnets: Compact and efficient convolutional neural networks via channel-wise convolutions. In *Advances in Neural Information Processing Systems*, pp. 5197–5205, 2018.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015b.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 243–254. IEEE, 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.

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, 2018.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

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.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Xiaojie Jin, Yingzhen Yang, Ning Xu, Jianchao Yang, Nebojsa Jojic, Jiashi Feng, and Shuicheng Yan. Wsnet: Compact and efficient networks through weight sampling. *arXiv preprint arXiv:1711.10067*, 2017.

Nebojsa Jojic, Brendan J Frey, and Anitha Kannan. Epitomic analysis of appearance and shape. In *ICCV*, volume 3, pp. 34, 2003.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Xin Li, Yiming Zhou, Zheng Pan, and Jiashi Feng. Partial order pruning: for best speed/accuracy trade-off in neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9145–9153, 2019.

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

Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik-Manor. Asap: Architecture search, anneal and prune. *arXiv preprint arXiv:1904.04123*, 2019.

Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755*, 2016.

Karol J Piczak. Esc: Dataset for environmental sound classification. In *Proceedings of the 23rd ACM international conference on Multimedia*, pp. 1015–1018. ACM, 2015.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941*, 7, 2017.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pp. 525–542. Springer, 2016.

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.

Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *international conference on machine learning*, pp. 2217–2225, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Ke Sun, Mingjie Li, Dong Liu, and Jingdong Wang. Igcv3: Interleaved low-rank group convolutions for efficient deep neural networks. *arXiv preprint arXiv:1806.00178*, 2018.

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.

Paul Viola, Michael Jones, et al. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2001.

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:1812.03443v2*, 12 2018.

Jiahui Yu and Thomas Huang. Network slimming by slimmable networks: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019a.

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

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

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856, 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 Implementation details

A.1 MobileNetV2 Settings

Epitome dimensions for MobileNetV2 bottleneck. With our NES method, the shape of the feature map produced by each layer can be kept the same as the ones from the original model before compression. However, the number of channels in the feature map is reduced using the width multiplier method for MobileNetV2. Hence, for fair comparison, we only apply the width multiplier on the output dimension of the first 1×1 convolutional layer and the input channel dimension of the second 1×1 convolutional layer within the bottleneck blocks of MobileNetV2 for obtaining the same feature map shape between blocks as our method. Based on this principle, we generate weights tensor based on the epitome along the filter dimension for the first 1×1 convolutional layers within the bottleneck and along the input channel dimension for the second 1×1 convolutional layers.

Specifically, we set the epitome shape per layer as $(\#\text{in\_channels}, \frac{\#\text{out\_channels} \times \text{expansion}}{\text{multiplier}}, 1, k)$ for the first 1×1 convolution layer and $(\frac{\#\text{in\_channels} \times expansion}{\text{multiplier}}, \#\text{out\_channels}, 1, k)$ for the second 1×1 layer as shown in Table 5. Here, expansion is referring to the ratio between the input size of the bottleneck and the inner size as detailed in Figure 2 of (Sandler et al., 2018). The shape represents the number of input channels, the number of output channels and the kernel size, respectively. The compression ratio for each layer, c, can thus be calculated as $c = \frac{1}{multiplier}$.

Table 5: Epitome dimensions for the inverted residual blocks of the MobileNetV2 backbone. Here w, h, k, k' denotes the spatial size, input channels and output channels of the input feature map respectively. Variables w_c and h_c denote the spatial size of the epitome and are set to 1 for 1×1 convolutional layer. c is used to set the compression ratio for each layer and is similar to the concept of width multiplier as defined in MobileNet (Howard et al., 2017). t is the expansion ratio as defined in MobileNetV2.

Input	Operators	Output	Epitome	Comp. ratio
$h \times w \times k$	1×1, conv2d, ReLU6	$h \times w \times tk$	$w_c \times h_c \times k \times ctk$	$\frac{w_c \times h_c}{c}$
$\frac{h}{s} \times \frac{w}{s} \times tk$	3×3, depth-wise separable, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times tk$	–	1
$\frac{h}{s} \times \frac{w}{s} \times tk$	1×1, conv2d, linear	$\frac{h}{s} \times \frac{w}{s} \times k'$	$w_c \times h_c \times ctk \times k'$	$\frac{w_c \times h_c}{c}$

A.2 Epitome Dimension Design

The size of the epitome can be calculated precisely by the original model and the desired compression ratio r. For a CNN with C n-dimensional convolutional layers and K fully-connected layers, its number of parameters can be calculated as $\sum_{i=1}^{C} \prod_d L_d^i + \sum_{k=1}^{K} N_{in}^k N_{out}^k$, where L_d^i denotes the length of the convolution weight tensor along the d^{th} dimension of the i^{th} convolutional layer. N_{in}^k and N_{out}^k denote the input and output dimension of the k^{th} fully-connected layer.

We assign an epitome $E^j \in \mathbb{R}^{W_j' \times H_j'}$ for each layer j, and a routing map $\mathcal{M} : (x_1, x_2, \ldots, x_n) \rightarrow (p, q)$, *i.e.*, the weight value of an n-d filter at location $(x_1, x_2, \ldots, x_n)$ being equal to $E(p, q)$. We define the dimension of the epitome to be 2D here to illustrate the general case and later, we will show that in practice, the dimension of the epitome can be increased to save the computation memory. The size of the total epitome is, thus, $\sum_{j=1}^{C+K} W_j' \times H_j'$. After learning the routing map for layer j, we store the location mapping as a lookup table of size M_j. The size of all the mapping tables is $\sum_{j=1}^{C+K} M_j$. Hence, the compression ratio can be calculated as

$$r = \frac{\sum_i^C \prod_d L_i^d + \sum_k^K N_{in}^k N_{out}^k}{\sum_j^{C+K} (W_j' \times H_j' + M_j)}. \tag{8}$$

In our analysis, we use a uniform compression ratio for all the layers. Therefore, given a compression ratio r, the size of the epitome for each layer can be calculated accordingly. This deterministic design of the epitome size is hardware friendly and can be used to control the memory allocation.

B More Results on 1D Convolution Compression

We also conduct experiments to examine the highest compression ratio that our method can achieve without performance drop compared to WSNet (Jin et al., 2017). For fair comparison, we choose the same 8-layer CNN model backbone as used in WSNet. Configuration details have been demonstrated in Table 1 in the formal paper. The results are shown in Table 6.

Table 6: Comparison with WSNet on ESC-50 dataset. We choose the compressed model by WSNet method as our baseline. By decreasing the size of the epitome, we can achieve higher compression ratio. We apply uniform compression ratio for all layers. It is observed that the highest compression ratio we can achieve before our method's performance become smaller than WSNet is $3.16\times$.

Methods	Compression Rate	Accuracy (%) (top1)
WSNet	$1.00\times$	66.5
Our method-1	$1\times$	73.0
Our method-2	$2.35\times$	69.25
Our method-3	$3.16\times$	65.5

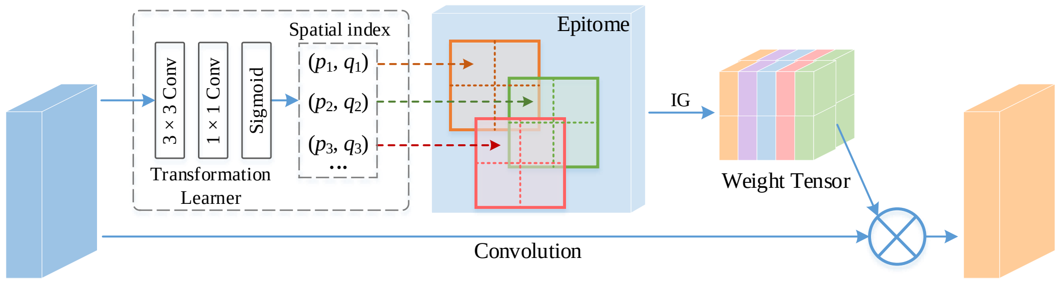

Figure 4: Our proposed compression process. We only show the transformation along spatial dimensions for easy understanding. The transformation learner learns the position mapping function, $\mathcal{M} : (i, j) \rightarrow (p, q)$, between the epitome element $E_{(p,q)}$ and the convolution kernel elements. Selected portions in the epitome will be concatenated together to form the weight tensor. Note that we use a moving average way to update $\mathcal{M}$ so that the transformation learner can be removed during inference. The whole training process is end-to-end trainable and hence can be used for any specified network architecture. 'IG' in the figure denotes the interpolation-based generator (Eqn. 3).

C NES for Fully-Connected Layer

Let $D \in \mathbb{R}^{N_{in} \times N_{out}}$ denotes the parameter matrix for a fully-connected (FC) layer. We could use Eqn. (10) to sampling along the input dimension and Eqn. (15) to sample along the output dimension. We also conduct experiments on CIFAR-10 dataset to verify the efficacy of our method on FC layers as shown in Table 7. We take MobileNetV2-1.0 as our baseline.

Table 7: Experiment results of our method applied on fully connected layer of MobileNetV2.

Methods	Parameters	Model Comp. Rate	FC Param. Comp. Rate	Top-1 Acc. (%)
MobileNetV2-1	2.2M	$1.00\times$	$1.00\times$	94.06
MobileNetV2-0.5 FC*	0.4 M	$5.55\times$	$2.00\times$	91.8
Our method-0.5 FC	0.39M	$5.64\times$	$2.00\times$	**92.96**

D Proof on Parameter and Computation Reduction

With the above defined compact epitome E, our method introduces a novel transformation function where the convolution filter weights are transformed from E with $\tau(\cdot)$. To simplify the illustration, we particularly consider the 2D convolution as an example. Our method can be extended to n-dimension convolution and fully-connected layers straightforwardly.

In our method, all the weight parameters $\mathcal{K}_{i,j,m,c}$ are transformed from the compact epitome E.

$$G_{t_w,t_h,c} = f_{t_w,t_h} * k_c = \sum_{i}^{W} \sum_{j}^{H} \sum_{m}^{C_{in}} F_{t_w+i,t_h+j,m} E_{\mathcal{M}(i,j,m,c)}. \tag{9}$$

Proof on computational cost reduction. Since the weight elements per convolutional layer are formed based on the same E, there is computational redundancy when two convolution kernels are selected from the

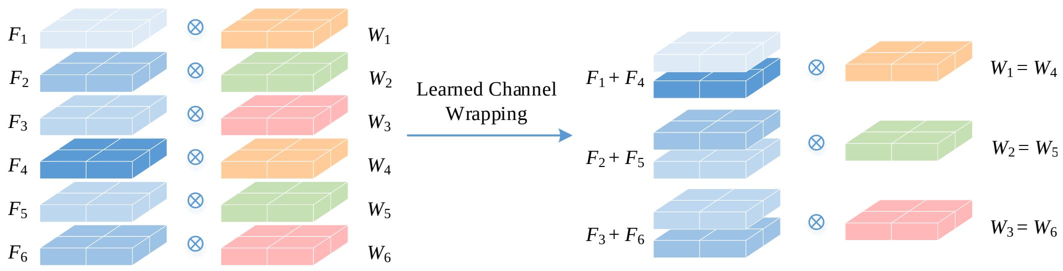

Figure 5: NES transformation along the input channel dimension. The input feature map $\mathcal{F}$ is first grouped based on the learned interpolation position of the kernels. Input feature map $\mathcal{F}_1$ and $\mathcal{F}_4$ are both multiplied with the weight kernel W_1. To reuse the multiplication, feature map $\mathcal{F}_1$ and $\mathcal{F}_4$ are first added together before multiplying with the weights kernel W_1.

same portion of E as shown in Figure 4. To reuse the multiplication in convolution, we first compute the multiplication between each element in the epitome E and the input feature map. The results are saved as a *product map* P such that the computations are done only once. However, given the epitome $E \in \mathbb{R}^{W' \times H'}$, the product map size is $W \times H \times C_{in} \times W' \times H' \times C_{out}$ which consumes large computational memory. To reduce its size, we propose to increase the dimensions of the epitome to $E' \in \mathbb{R}^{W' \times H' \times C'_{in} \times C'_{out}}$ in order to group the computation results in the product map. Each entry in the product map P is calculated as the dot product between the channel dimension along the input feature map and the third dimension along the compact weight matrix. We set C'_{in} to be smaller than C_{in} to further boost the compression and β_1 equal to C'_{in}. As illustrated in Figure 5, the input channels of the feature map is first grouped by

$$\tilde{F}(i,j,k) = \sum_{c=0}^{R_{cin}} F(p,q,k + c \times C'_{in} + c'), \tag{10}$$

where $R_{cin} = \lceil C_{in}/C'_{in} \rceil$ is the compression ratio along the input channel dimension and $k + c' + c \times C'_{in}$ is the learned position with $(p,q,c',n) = \mathcal{M}(i,j,c,m)$, where $c' \in [1, C'_{in}]$. The transformed input feature map $\tilde{F}$ has the same number of channels as C'_{in}. Let (i,j,k) index the transformed feature map location. The product map is then calculated as

$$P_{i,j,p,q,n} = \tilde{\mathcal{F}}_{i,j,:} \cdot E'_{p,q,:,n}, \tag{11}$$

where $\cdot$ denotes the dot product operation.

Now, the multiplications can be reused by replacing the convolution kernels with the product map P. Based on Eqns. (9), (10), (11), the convolution can be reduced by

$$G_{t_w,t_h,n} = \sum_{i}^{W} \sum_{j}^{H} P_{(t_w+i,t_h+j,p,q,n)}. \tag{12}$$

To reuse the additions, we adopt an integral image I which is proposed in (Crow, 1984) but differently we extend the integral image dimension to make it suitable for 2D convolution based on P. Our integral image can be constructed by

$$I(t_w,t_h,p,q,n) = \begin{cases} P_{0,t_h,p,q,n}, & t_w = 0 \\ P_{t_w,0,p,q,n}, & t_h = 0 \\ P_{t_w,t_h,0,q,n}, & p = 0 \\ P_{t_w,t_h,p,0,n}, & q = 0 \\ P_{t_w,t_h,p,q,0}, & n = 0 \\ I(t_w-1,t_h-1,p-1,q-1,n-1) + P(t_w,t_h,p,q,n), & \text{else.} \end{cases} \tag{13}$$

From Eqn. 13, the 2D convolution results can be retrieved in a similar way to (Jin et al., 2017) as follows:

$$G_{t_w,t_h,p,q,n} = I(t_w+w-1,t_h+h-1,p+w-1,q+h-1,n) - I(t_w-1,t_h-1,p-1,q-1,n-1). \tag{14}$$

As we set C'_{out} to be smaller than the output channel dimension of the convolution kernel, we reuse the computation results from the epitome via

$$\tilde{G}_{t_w,t_h,r_{out}\times\beta_2:(r_{out}+1)\times\beta_2} = G_{t_w,t_h,n:n+\beta_2}, \tag{15}$$

where $r_{out} \in \{0, 1, ..., R_{cout} - 1\}$, $R_{cout} = \lceil C_{out}/\beta_2 \rceil$ is the number of transformations conducted along the output channel dimension, and $n = \mathcal{M}(r_{out} \times \beta_2)$ is the learned mapping. The filter length $\beta_2 \in \{1, 2, ..., C'_{out}\}$ is a hyper-parameter, which is $C'_{out}/2$. Thus, the MAdds can be calculated as

$$\text{Reduced MAdd} = \underbrace{(2C_{in}W_cH_c - 1)WHC'_{out}}_{\text{From Eqn. 11}} + \underbrace{WHW_cH_cC'_{out}}_{\text{From Eqn. 13}} + \underbrace{2R_{cin}\beta_1}_{\text{From Eqn. 10}} + \underbrace{2R_{cout}\beta_2}_{\text{From Eqn.15}}. \tag{16}$$

Suppose we use sliding window with a stride of 1 and no bias term, the MAdds of the conventional convolution can be calculated based on Eqn. 2:

$$\text{MAdd} = (2 \times C_{in} \times w \times h - 1) \times H \times W \times C_{out}. \tag{17}$$

Therefore, the total MAdd reduction ratio is

$$\text{MAdd Reduction Ratio} = \frac{C_{out}HW(2C_{in}wh - 1)}{C'_{out}HW(W_cH_c + 2C'_{in}W_cH_c - 1) + 2R_{cin}\beta_1 + 2R_{cout}\beta_2}. \tag{18}$$

Proof on parameter reduction. The parameter compression ratio for a 2D convolution layer can be calculated via Eqn. 8 as follows:

$$r = \frac{whC_{in}C_{out}}{W_c \times H_c \times C'_{in} \times C'_{out} + 3 \times R_{cin} + R_{cout}}, \tag{19}$$

where w and h denote the width and height of the kernel of the layer. W_c and H_c denote the spatial size of the corresponding epitome. From Eqn. 18 and Eqn. 19, it can be observed that the compression ratio is mainly decided by $\frac{C_{out}C_{in}wh}{C'_{out}C'_{in}W_cH_c}$.