Dynamic Convolution: Attention over Convolution Kernels

Yinpeng Chen Xiyang Dai Mengchen Liu Dongdong Chen Lu Yuan Zicheng Liu
Microsoft
{yiche,xidai,mengcliu,dochen,luyuan,zliu}@microsoft.com

Abstract

Light-weight convolutional neural networks (CNNs) suffer performance degradation as their low computational budgets constrain both the depth (number of convolution layers) and the width (number of channels) of CNNs, resulting in limited representation capability. To address this issue, we present Dynamic Convolution, a new design that increases model complexity without increasing the network depth or width. Instead of using a single convolution kernel per layer, dynamic convolution aggregates multiple parallel convolution kernels dynamically based upon their attentions, which are input dependent. Assembling multiple kernels is not only computationally efficient due to the small kernel size, but also has more representation power since these kernels are aggregated in a non-linear way via attention. By simply using dynamic convolution for the state-of-the-art architecture MobileNetV3-Small, the top-1 accuracy of ImageNet classification is boosted by 2.9% with only 4% additional FLOPs and 2.9 AP gain is achieved on COCO keypoint detection.

1. Introduction

Interest in building light-weight and efficient neural networks has exploded recently. It not only enables new experiences on mobile devices, but also protects user's privacy from sending personal information to the cloud. Recent works (e.g. MobileNet [12, 27, 11] and ShuffleNet [42, 23]) have shown that both efficient operator design (e.g. depthwise convolution, channel shuffle, squeeze-and-excitation [13], asymmetric convolution [5]) and architecture search ([29, 7, 2]) are important for designing efficient convolutional neural networks.

However, even the state-of-the-art efficient CNNs (e.g. MobileNetV3 [11]) suffer significant performance degradation when the computational constraint becomes extremely low. For instance, when the computational cost of MobileNetV3 reduces from 219M to 66M Multi-Adds, the top-1 accuracy of ImageNet classification drops from 75.2% to 67.4%. This is because the extremely low computational cost severely constrains both the network depth (number of layers) and width (number of channels), which are crucial for the network performance but proportional to the computational cost.

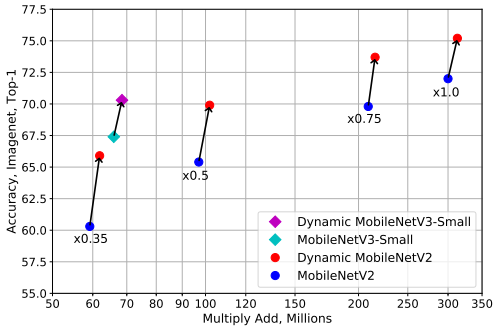

Figure 1. The trade-off between computational cost (MAdds) and top-1 accuracy of ImageNet classification. Dynamic convolution significantly boosts the accuracy with a small amount of extra MAdds on MobileNet V2 and V3. Best viewed in color.

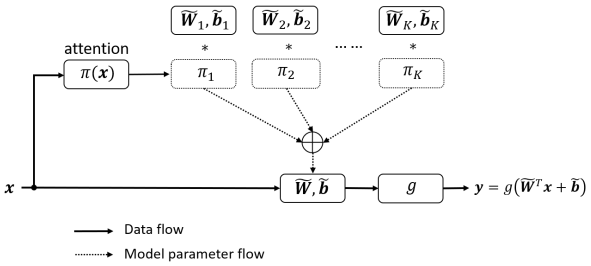

Figure 2. Dynamic perceptron. It aggregates multiple linear functions dynamically based upon their attentions $\{\pi_k\}$, which are input dependent.

This paper proposes a new operator design, named *dynamic convolution*, to increase the representation capability with negligible extra FLOPs. Dynamic convolution uses a set of K parallel convolution kernels $\{\tilde{\boldsymbol{W}}_k, \tilde{\boldsymbol{b}}_k\}$ instead of using a single convolution kernel per layer (see Figure 2). These convolution kernels are aggregated dynamically $\tilde{\boldsymbol{W}} = \sum_k \pi_k(\boldsymbol{x})\tilde{\boldsymbol{W}}_k$ for each individual input $\boldsymbol{x}$ (e.g. im-

age) via input dependent attention $\pi_k(\boldsymbol{x})$. The biases are aggregated using the same attention $\tilde{\boldsymbol{b}} = \sum_k \pi_k(\boldsymbol{x})\tilde{\boldsymbol{b}}_k$. Dynamic convolution is a non-linear function with more representation power than its static counterpart. Meanwhile, dynamic convolution is computationally efficient. It does not increase the depth or width of the network, as the parallel convolution kernels share the output channels by aggregation. It only introduces extra computational cost to compute attentions $\{\pi_k(\boldsymbol{x})\}$ and aggregate kernels , which is negligible compared to convolution. *The key insight is that within reasonable cost of model size (as convolution kernels are small), dynamic kernel aggregation provides an efficient way (low extra FLOPs) to boost representation capability.*

Dynamic convolutional neural networks (denoted as DY-CNNs) are more difficult to train, as they require joint optimization of all convolution kernels and the attention across multiple layers. We found two keys for efficient joint optimization: (a) constraining the attention output as $\sum_k \pi_k(\boldsymbol{x}) = 1$ to facilitate the learning of attention model $\pi_k(\boldsymbol{x})$, and (b) flattening attention (near-uniform) in early training epochs to facilitate the learning of convolution kernels $\{\tilde{\boldsymbol{W}}_k, \tilde{\boldsymbol{b}}_k\}$. We simply integrate these two keys by using softmax with a large temperature for kernel attention.

We demonstrate the effectiveness of dynamic convolution on both image classification (ImageNet) and keypoint detection (COCO). Without bells and whistles, simply replacing static convolution with dynamic convolution in MobileNet V2 and V3 achieves solid improvement with only a slight increase (4%) of computational cost (see Figure 1). For instance, with 100M Multi-Adds budget, our method gains 4.5% and 2.9% top-1 accuracy on image classification for MobileNetV2 and MobileNetV3, respectively.

2. Related Work

Efficient CNNs: Recently, designing efficient CNN architectures [15, 12, 27, 11, 42, 23] has been an active research area. SqueezeNet [15] reduces the number of parameters by using 1×1 convolution extensively in the fire module. MobileNetV1 [12] substantially reduces FLOPs by decomposing a 3×3 convolution into a depthwise convolution and a pointwise convolution. Based upon this, MobileNetV2 [27] introduces inverted residuals and linear bottlenecks. MobileNetV3 [11] applies squeeze-and-excitation [13] in the residual layer and employs a platform-aware neural architecture approach [29] to find the optimal network structures. ShuffleNet further reduces MAdds for 1×1 convolution by channel shuffle operations. ShiftNet [33] replaces expensive spatial convolution by the shift operation and pointwise convolutions. Compared with existing methods, our dynamic convolution can be used to replace any static convolution kernels (e.g. 1×1, 3×3, depthwise convolution, group convolution) and is complementary to other advanced operators like squeeze-and-excitation.

Model Compression and Quantization: Model compression [8, 22, 10] and quantization [3, 43, 40, 38, 30] approaches are also important for learning efficient neural networks. They are complementary to our work, helping reduce the model size for our dynamic convolution method.

Dynamic Deep Neural Networks: Our method is related to recent works of dynamic neural networks [18, 21, 31, 34, 39, 14] that focus on skipping part of an existing model based on input image. D^{2}NN [21], SkipNet [31] and BlockDrop [34] learn an additional controller for skipping decision by using reinforcement learning. MSDNet [14] allows early-exit based on the current prediction confidence. Slimmable Nets [39] learns a single neural network executable at different width. Once-for-all [1] proposes a progressive shrinking algorithm to train one network that supports multiple sub-networks. The accuracy for these sub-networks is the same as independently trained networks. Compared with these works, our method has two major differences. Firstly, our method has dynamic convolution kernels but static network structure, while existing works have static convolution kernels but dynamic network structure. Secondly, our method does *not* require an additional controller. The attention is embedded in each layer, enabling end-to-end training. Compared to the concurrent work [37], our method is more efficient with better performance.

Neural Architecture Search: Recent research works in neural architecture search (NAS) are powerful on finding high-accuracy neural network architectures [44, 26, 45, 20, 36] as well as hardware-aware efficient network architectures [2, 29, 32]. The hardware-aware NAS methods incorporate hardware latency into the architecture search process, by making it differentiable. [7] proposed single path supernet to optimize all architectures in the search space simultaneously, and then perform evolutionary architecture search to handle computational constraints. Based upon NAS, MobileNetV3 [11] shows significant improvements over human-designed baselines (e.g. MobileNetV2 [27]). Our dynamic convolution method can be easily used in advanced architectures found by NAS. Later in this paper, we will show that dynamic convolution not only improves the performance for human-designed networks (e.g. MobileNetV2), but also boosts the performance for automatically searched architectures (e.g. MobileNetV3), with low extra FLOPs. In addition, our method provides a new and effective component to enrich the search space.

3. Dynamic Convolutional Neural Networks

We describe dynamic convolutional neural networks (DY-CNNs) in this section. The goal is to provide better trade-off between network performance and computational burden, within the scope of efficient neural networks. The two most popular strategies to boost the performance are making neural networks "deeper" or "wider". How-

ever, they both incur heavy computation cost, thus are not friendly to efficient neural networks.

We propose dynamic convolution, which does not increase either the depth or the width of the network, but increase the model capability by aggregating multiple convolution kernels via attention. Note that these kernels are assembled differently for different input images, from where *dynamic convolution* gets its name. In this section, We firstly define the generic dynamic perceptron, and then apply it to convolution.

3.1. Preliminary: Dynamic Perceptron

Definition: Let us denote the traditional or static perceptron as $\boldsymbol{y} = g(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})$, where $\boldsymbol{W}$ and $\boldsymbol{b}$ are weight matrix and bias vector, and g is an activation function (e.g. ReLU [24, 16]). We define the dynamic perceptron by aggregating multiple (K) linear functions $\{\tilde{\boldsymbol{W}}_k^T\boldsymbol{x} + \tilde{\boldsymbol{b}}_k\}$ as follows:

$$\begin{aligned} \boldsymbol{y} &= g(\tilde{\boldsymbol{W}}^T(\boldsymbol{x})\boldsymbol{x} + \tilde{\boldsymbol{b}}(\boldsymbol{x})) \\ \tilde{\boldsymbol{W}}(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k(\boldsymbol{x})\tilde{\boldsymbol{W}}_k,\ &\tilde{\boldsymbol{b}}(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k(\boldsymbol{x})\tilde{\boldsymbol{b}}_k \\ \text{s.t.}\quad 0 \le \pi_k(\boldsymbol{x}) \le 1, &\sum_{k=1}^{K} \pi_k(\boldsymbol{x}) = 1, \end{aligned} \tag{1}$$

where π_k is the attention weight for the k^{th} linear function $\tilde{\boldsymbol{W}}_k^T\boldsymbol{x}+\tilde{\boldsymbol{b}}_k$. Note that the aggregated weight $\tilde{\boldsymbol{W}}(\boldsymbol{x})$ and bias $\tilde{\boldsymbol{b}}(\boldsymbol{x})$ are functions of input and share the same attention.

Attention: the attention weights $\{\pi_k(\boldsymbol{x})\}$ are not fixed, but vary for each input $\boldsymbol{x}$. They represent the optimal aggregation of linear models $\{\tilde{\boldsymbol{W}}_k^T\boldsymbol{x} + \tilde{\boldsymbol{b}}_k\}$ for a given input. The aggregated model $\tilde{\boldsymbol{W}}^T(\boldsymbol{x})\boldsymbol{x}+\tilde{\boldsymbol{b}}(\boldsymbol{x})$ is a non-linear function. Thus, dynamic perceptron has more representation power than its static counterpart.

Computational Constraint: compared with static perceptron, dynamic perceptron has the same number of output channels but bigger model size. It also introduces two additional computations: (a) computing the attention weights $\{\pi_k(\boldsymbol{x})\}$, and (b) aggregating parameters based upon attention $\sum_k \pi_k\tilde{\boldsymbol{W}}_k$ and $\sum_k \pi_k\tilde{\boldsymbol{b}}_k$. The additional computational cost should be significantly less than the cost of computing $\tilde{\boldsymbol{W}}^T\boldsymbol{x} + \tilde{\boldsymbol{b}}$. Mathematically, the computational constraint can be represented as follows:

$$\begin{aligned} O(\tilde{\boldsymbol{W}}^T\boldsymbol{x} + \tilde{\boldsymbol{b}}) \gg O\left(\sum \pi_k\tilde{\boldsymbol{W}}_k\right) + O\left(\sum \pi_k\tilde{\boldsymbol{b}}_k\right) \\ + O\left(\pi(\boldsymbol{x})\right) \end{aligned} \tag{2}$$

where $O(\cdot)$ measures the computational cost (e.g. FLOPs). Note that fully connected layer does not satisfy this, while convolution is a proper fit for this constraint.

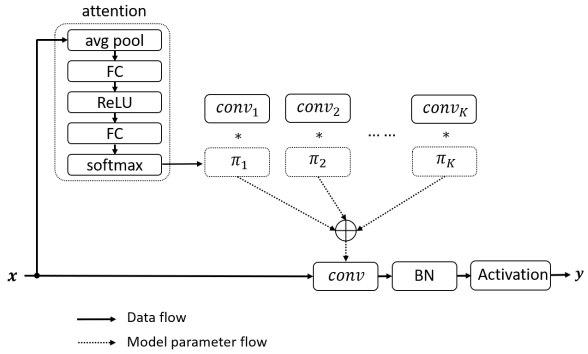

Figure 3. A dynamic convolution layer.

3.2. Dynamic Convolution

In this subsection, we showcase a specific dynamic perceptron, dynamic convolution that satisfies the computational constraint (Eq. 2). Similar to dynamic perceptron, dynamic convolution (Figure 3) has K convolution kernels that share the same kernel size and input/output dimensions. They are aggregated by using the attention weights $\{\pi_k\}$. Following the classic design in CNN, we use batch normalization and an activation function (e.g. ReLU) after the aggregated convolution to build a dynamic convolution layer.

Attention: we apply squeeze-and-excitation [13] to compute kernel attentions $\{\pi_k(\boldsymbol{x})\}$ (see Figure 3). The global spatial information is firstly squeezed by global average pooling. Then we use two fully connected layers (with a ReLU between them) and softmax to generate normalized attention weights for K convolution kernels. The first fully connected layer reduces the dimension by 4. Different from SENet [13] which computes attentions over output channels, we compute attentions over convolution kernels. The computation cost for the attention is cheap. For an input feature map with dimension $H \times W \times C_{in}$, the attention requires $O(\pi(\boldsymbol{x})) = HWC_{in}+C_{in}^2/4+C_{in}K/4$ Mult-Adds. This is much less than the computational cost of convolution, i.e. $O(\tilde{\boldsymbol{W}}^T\boldsymbol{x} + \tilde{\boldsymbol{b}}) = HWC_{in}C_{out}D_k^2$ Mult-Adds, where D_k is the kernel size, and C_{out} is the number of output channels.

Kernel Aggregation: aggregating convolution kernels is computationally efficient due to the small kernel size. Aggregating K convolution kernels with kernel size $D_k \times D_k$, C_{in} input channels and C_{out} output channels introduces $KC_{in}C_{out}D_k^2 + KC_{out}$ extra Multi-Adds. Compared with the computational cost of convolution ($HWC_{in}C_{out}D_k^2$), the extra cost is neligible if $K \ll HW$. Table 1 shows the computational cost of using dynamic convolution in MobileNetV2. For instance, when using MobileNetV2 ($\times$1.0), dynamic convolution with $K = 4$ kernels only increases the computation cost by 4%. Note that even though dynamic convolution increases the model size, it does not increase

	×1.0	×0.75	×0.5	×0.35
static	300.0M	209.0M	97.0M	59.2M
K=2	309.5M	215.6M	100.5M	61.5M
K=4	312.9M	217.5M	101.4M	62.0M
K=6	316.3M	219.5M	102.3M	62.5M
K=8	319.8M	221.4M	103.2M	62.9M

Table 1. Mult-Adds of static convolution and dynamic convolution in MobileNetV2 with four different width multipliers ($\times 1.0$, $\times 0.75$, $\times 0.5$, and $\times 0.35$).

the output dimension of each layer. The amount of the increase is acceptable as convolution kernels are small.

From CNNs to DY-CNNs: dynamic convolution can be easily used as a drop-in replacement for any convolution (e.g. 1×1 conv, 3×3 conv, group convolution, depth-wise convolution) in any CNN architecture. It is also complementary to other operators (like squeeze-and-excitation [13]) and activation functions (e.g. ReLU6, h-swish [11]). In the rest of the paper, we use prefix **DY-** for the networks that use dynamic convolution. For example, DY-MobileNetV2 refers to using dynamic convolution in MobileNetV2. We also use weight $\tilde{\boldsymbol{W}}_k$ to denote a convolution kernel and ignore bias $\tilde{\boldsymbol{b}}_k$, for the sake of brevity.

4. Two Insights of Training Deep DY-CNNs

Training deep DY-CNNs is challenging, as it requires joint optimization of all convolution kernels $\{\tilde{\boldsymbol{W}}_k\}$ and attention model $\pi_k(\boldsymbol{x})$ across multiple layers. In this section, we discuss two insights for more *efficient joint optimization*, which are crucial especially to deep DY-CNNs.

4.1. Insight 1: Sum the Attention to One

The first insight is: *constraining the attention output can facilitate the learning of attention model* $\pi_k(\boldsymbol{x})$. Specifically, we have the constraint $\sum_k \pi_k(\boldsymbol{x}) = 1$ in Eq. (1) to keep the aggregated kernel $\tilde{\boldsymbol{W}} = \sum_k \pi_k \tilde{\boldsymbol{W}}_k$ within the convex hull of $\{\tilde{\boldsymbol{W}}_k\}$ in the kernel space. Figure 4 shows an example with 3 convolution kernels. The constraint $0 \leq \pi_k(\boldsymbol{x}) \leq 1$ only keeps the aggregated kernel within the two pyramids. The sum-to-one constraint further compresses the kernel space to a triangle. It compresses the red line that comes from the origin into a dot by normalizing the attention sum. This normalization significantly simplifies the learning of $\pi_k(\boldsymbol{x})$, when it is jointly optimized with $\{\tilde{\boldsymbol{W}}_k\}$ in a deep network. Softmax is a natural choice of $\sum_k \pi_k(\boldsymbol{x}) = 1$.

4.2. Insight 2: Near-uniform Attention in Early Training Epochs

The second insight is: *near-uniform attention can facilitate the learning of all kernels* $\{\tilde{\boldsymbol{W}}_k\}$ *during early training epochs*. This is because near-uniform attention enables more convolution kernels to be optimized simultaneously.

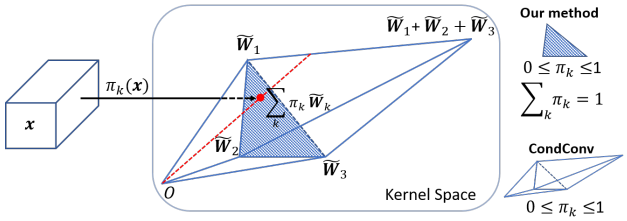

Figure 4. Illustration of constraint $\sum_k \pi_k(\boldsymbol{x}) = 1$. It compresses the space of aggregated kernel $\sum_k \pi_k \tilde{\boldsymbol{W}}_k$ from two pyramids (used in CondConv [37]) to a triangle. A red line is compressed into a dot by normalizing attention sum. Best viewed in color.

Softmax does NOT work well on this due to its near one-hot output. It only allows a small subset of kernels across layers to be optimized. Figure 5-(Left) shows that the training converges slowly when using softmax (blue curves) to compute attention. Here, DY-MobileNetV2 with width multiplier $\times 0.5$ is used. The final top-1 accuracy (64.8%) is even worse than its static counterpart (65.4%). This inefficiency is related to the number of dynamic convolution layers. To validate this, we reduce the number of dynamic convolution layers by 3 (only use dynamic convolution for the last 1×1 convolution in each bottleneck residual block) and expect faster convergence in training. The training and validation errors are shown in Figure 5-(Right) (blue curves). As we expected, the training converges faster with higher top-1 accuracy (65.9%) at the end.

We address this inefficiency in training deeper DY-CNNs by using a large temperature in softmax to flatten attention as follows:

$$\pi_k = \frac{\exp(z_k/\tau)}{\sum_j \exp(z_j/\tau)}, \tag{3}$$

where z_k is the output of the second FC layer in attention branch (see Figure 3), and τ is the temperature. The original softmax is a special case ($\tau = 1$). As τ increases, the output is less sparse. When using a large temperature $\tau = 30$, the training becomes significantly more efficient (see the red curves in Figure 5-(Left)). As a result, the top-1 accuracy boosts to 69.4%. The larger temperature is also helpful when stacking fewer dynamic convolution layers (see red curves in Figure 5-(Right)).

Temperature annealing, i.e. reducing τ from 30 to 1 linearly in the first 10 epochs, can further improve the top-1 accuracy (from 69.4% to 69.9%). These results support that near-uniform attention in early training epochs is crucial.

4.3. Relation to Concurrent Work

These two insights are the key differences between our method and the concurrent work (CondConv [37]), which uses sigmoid to compute kernel attention. Even though sigmoid provides near-uniform attention in early training

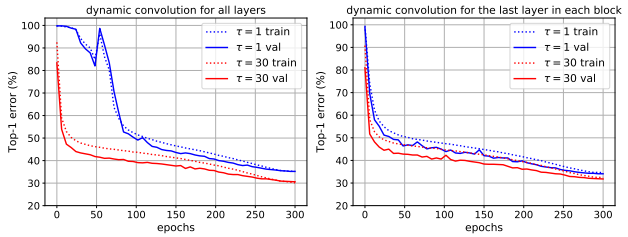

Figure 5. Training and validation errors for using different softmax temperatures. **Left**: using dynamic convolution for all layers. **Right**: using dynamic convolution for the last layer in each bottleneck residual block. We use DY-MobileNetV2 with width multiplier $\times 0.5$, and each dynamic convolution layer has $K = 4$ convolution kernels. Best viewed in color.

	Method	#Kernels	#Param	MAdds	Top-1
$\times 1.0$	CondConv [37]	8	27.5M	329M	74.6
	DY-CNNs (ours)	4	11.1M	312.9M	**75.2**
$\times 0.5$	CondConv [37]	8	15.5M	113M	68.4
	DY-CNNs (ours)	4	4M	101.4M	**69.9**

Table 2. Comparison between DY-CNNs and the concurrent work (CondConv [37]) on ImageNet classification using MobileNetV2 $\times 1.0$ and $\times 0.5$.

epochs, it has significantly larger kernel space (two pyramids in Figure 4) than our method (the shaded triangle in Figure 4). Thus, learning attention model $\pi_k(\boldsymbol{x})$ becomes more difficult. As a result, our method has less kernels per layer, smaller model size, less computations but achieves higher accuracy (see Table 2).

5. Experiments: ImageNet Classification

In this section, we present experimental results of dynamic convolution along with comprehensive ablations on ImageNet [4] classification. ImageNet has 1000 classes, including 1,281,167 images for training and 50,000 images for validation.

5.1. Implementation Details

We evaluate dynamic convolution on three architectures (ResNet [9], MobileNetV2[27], and MobileNetV3 [11]), by using dynamic convolution for all convolution layers except the first layer. Each layer has $K = 4$ convolution kernels. The batch size is 256. We use different training setups for the three architectures as follows:

Training setup for DY-ResNet: The initial learning rate is 0.1 and drops by 10 at epoch 30, 60 and 90. The weight decay is 1e-4. All models are trained using SGD optimizer with 0.9 momentum for 100 epochs. We use dropout rate 0.1 before the last layer of DY-ResNet-18.

Training setup for DY-MobileNetV2: The initial learning rate is 0.05 and is scheduled to arrive at zero within a single cosine cycle. The weight decay is 4e-5. All models are trained using SGD optimizer with 0.9 momentum for 300 epochs. To prevent overfitting, we use label smoothing and dropout before the last layer for larger width multipliers ($\times 1.0$ and $\times 0.75$). The dropout rate are 0.2 and 0.1 for $\times 1.0$ and $\times 0.75$, respectively. Mixup [41] is used for $\times 1.0$.

Training setup for DY-MobileNetV3: The initial learning rate is 0.1 and is scheduled to arrive at zero within a single cosine cycle. The weight decay is 3e-5. We use SGD optimizer with 0.9 momentum for 300 epochs and dropout rate of 0.2 before the last layer.

Kernel Aggregation	Top-1	Top-5
attention: $\sum \pi_k(\boldsymbol{x})\tilde{\boldsymbol{W}}_k$	69.4	88.6
average: $\sum \tilde{\boldsymbol{W}}_k/K$	36.0	61.5
max: $\tilde{\boldsymbol{W}}_{\text{argmax}_k(\pi_k)}$	0.1	0.5
shuffle per image: $\sum \pi_j(\boldsymbol{x})\tilde{\boldsymbol{W}}_k, j \neq k$	14.8	30.5
shuffle across images: $\left(\sum \pi_k(\boldsymbol{x})\tilde{\boldsymbol{W}}_k\right)(\boldsymbol{x}')$	27.3	48.4

Table 3. **Inspecting DY-CNN** using different kernel aggregations. DY-MobileNetV2 $\times 0.5$ is used. The proper aggregation of convolution kernels $\{\tilde{\boldsymbol{W}}_k\}$ using attention $\pi_k(\boldsymbol{x})$ is shown in the first line. Shuffle per image means shuffling the attention weights for the same image over different kernels. Shuffle across images means using the attention of an image $\boldsymbol{x}$ for another image $\boldsymbol{x}'$. The poor performance for the bottom four aggregations validates that the DY-CNN is dynamic.

5.2. Inspecting DY-CNN

We inspect if DY-CNN is dynamic, using DY-MobileNetV2 $\times 0.5$, which has $K = 4$ kernels per layer and is trained by using $\tau = 30$. Two properties are expected if it is dynamic: (a) *the convolution kernels are diverse per layer*, and (b) *the attention is input dependent*. We examine these two properties by contradiction. Firstly, if the convolution kernels are *not* diverse, the performances will be stable if different attentions are used. Thus, we vary the kernel aggregation per layer in three different ways: averaging $\sum \tilde{\boldsymbol{W}}_k/K$, choosing the convolution kernel with the maximum attention $\tilde{\boldsymbol{W}}_{\text{argmax}_k(\pi_k)}$, and random shuffling attention over kernels per image $\sum \pi_j(\boldsymbol{x})\tilde{\boldsymbol{W}}_k, j \neq k$. Compared with using the original attention, the performances of these variations are significantly degraded (shown in Table 3). When choosing the convolution kernel with the maximum attention, the top-1 accuracy (0.1) is as low as randomly choosing a class. The significant instability confirms the diversity of convolution kernels. In addition, we shuffle attentions across images to check if the attention is input dependent. The poor accuracy (27.3%) indicates that it is crucial for each image to use its own attention.

Furthermore, we inspect the attention across layers and find that attentions are flat at low levels and sparse at high levels. This is helpful to explain why variations in Table 3

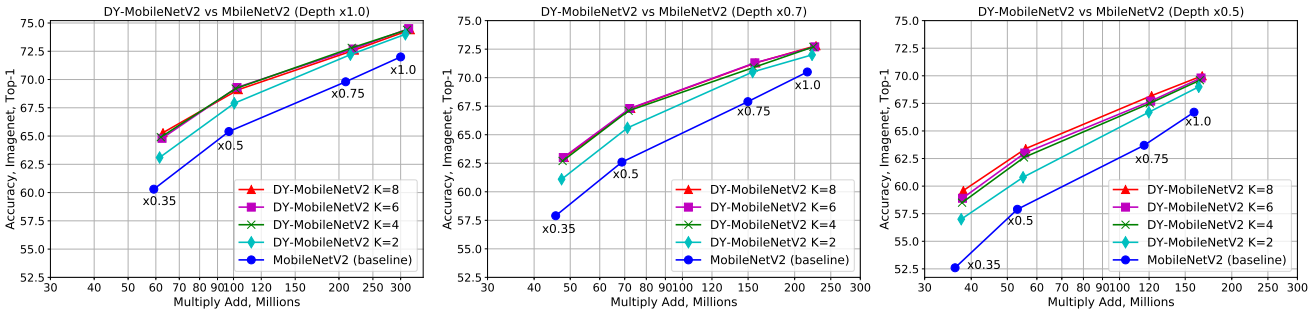

Figure 6. **The number of convolution kernels** (K) in DY-MobileNetV2 with different depth and width multipliers. **Left**: depth multiplier is 1.0, **Middle**: depth multiplier is 0.7, **Right**: depth multiplier is 0.5. Each curve has four width multipliers $\times 1.0$, $\times 0.75$, $\times 0.5$, and $\times 0.35$. Dynamic convolution outperforms its static counterpart by a clear margin for all width/depth multipliers. Best viewed in color.

Input Resolution						
112^2	56^2	28^2	14^2	7^2	Top-1	Top-5
–	–	–	–	✓	57.3	79.9
–	–	–	✓	✓	67.0	87.2
–	–	✓	✓	✓	67.5	87.4
–	✓	✓	✓	✓	69.1	88.4
✓	✓	✓	✓	✓	**69.4**	**88.6**
✓	✓	✓	✓	–	50.9	76.2
✓	✓	✓	–	–	42.5	68.4
✓	✓	–	–	–	41.2	67.0
✓	–	–	–	–	37.9	63.5
–	–	–	–	–	36.0	61.5

Table 4. **Inspecting DY-CNN** by enabling/disabling attention at different input resolutions. DY-MobileNetV2 $\times 0.5$ is used. Each resolution has two options: ✓indicates *enabling* attention $\sum \pi_k(\boldsymbol{x})\tilde{\boldsymbol{W}}_k$ for each layer in that resolution, while – indicates *disabling* attention and using average kernel $\sum \tilde{\boldsymbol{W}}_k/K$ for each layer in the corresponding resolutions. The attention is more effective at higher layers with lower resolutions.

Network	C1	C2	C3	Top-1	Top-5
MobileNetV2	1	1	1	65.4	86.4
DY-MobileNetV2	4	1	1	$67.4_{(2.0)}$	$87.5_{(1.1)}$
	1	4	1	$67.4_{(2.0)}$	$87.3_{(0.9)}$
	1	1	4	$68.2_{(2.8)}$	$87.9_{(1.5)}$
	4	1	4	$68.7_{(3.3)}$	$88.0_{(1.6)}$
	1	4	4	$68.4_{(3.0)}$	$87.9_{(1.5)}$
	4	4	1	$68.6_{(3.2)}$	$88.0_{(1.6)}$
	4	4	4	$\mathbf{69.4_{(4.0)}}$	$\mathbf{88.6_{(2.2)}}$

Table 5. **Dynamic convolution at different layers** in MobileNetV2 $\times 0.5$. C1, C2 and C3 indicate the 1×1 convolution that expands output channels, the 3×3 depthwise convolution and the 1×1 convolution that shrinks output channels per block respectively. C1=1 indicates using static convolution, while C1=4 indicates using dynamic convolution with 4 kernels. The numbers in brackets denote the improvement over the baseline.

have poor accuracy. For instance, averaging kernels with sparse attention at high levels or picking one convolution kernel (with the maximum attention) at low levels (where attention is flat) is problematic. Table 4 shows how attention affects the performance across layers. We group layers by their input resolutions, and switch on/off attention for these groups. If attention is switched off for a resolution, each layer in that resolution aggregates kernels by averaging. When enabling attention at higher levels alone (resolution 14^2 and 7^2), the top-1 accuracy is 67.0%, close to the performance (69.4%) of using attention for all layers. If attention is used for lower levels alone (resolution 112^2, 56^2 and 28^2), the top-1 accuracy is poor 42.5%.

5.3. Ablation Studies

We perform a number of ablations on DY-MobileNetV2 and DY-MobileNetV3. The default setup includes using $K = 4$ kernels per layer and $\tau = 30$.

The number of convolution kernels (K): the hyper-parameter K controls the model complexity. Figure 6 shows the classification accuracy and computational cost for dynamic convolution with different K. We compare DY-MobileNetV2 with MobileNetV2 on different depth/width multipliers. Firstly, the dynamic convolution outperforms its static counterpart for all depth/width multipliers, even with small $K = 2$. This demonstrates the strength of our method. In addition, the accuracy stops increasing once K is larger than 4. This is because as K increases, even though the model has more representation power, it is more difficult to optimize all convolution kernels and attention simultaneously and the network is more prone to over-fitting.

Dynamic convolution at different layers: Table 5 shows the classification accuracy for using dynamic convolution at three different layers (1×1 conv, 3×3 depthwise conv, 1×1 conv) per bottleneck residual block in MobileNetV2 $\times 0.5$. The accuracy increases as more dynamic convolution layers are used. Using dynamic convolution for all three layers yields the best accuracy. If only one layer is allowed to use dynamic convolution, using it for the last 1×1 convolution yields the best performance.

Network	Temperature	Top-1	Top-5
MobileNetV2	—	65.4	86.4
DY-MobileNetV2	$\tau = 1$	$64.8_{(-0.6)}$	$85.5_{(-0.9)}$
	$\tau = 5$	$65.7_{(+0.3)}$	$85.8_{(-0.6)}$
	$\tau = 10$	$67.5_{(+2.1)}$	$87.4_{(+1.0)}$
	$\tau = 20$	$\mathbf{69.4_{(+4.0)}}$	$88.5_{(+2.1)}$
	$\tau = 30$	$\mathbf{69.4_{(+4.0)}}$	$\mathbf{88.6_{(+2.2)}}$
	$\tau = 40$	$69.2_{(+3.8)}$	$88.4_{(+2.0)}$
	τ annealing	$\mathbf{69.9_{(+4.5)}}$	$\mathbf{89.0_{(+2.6)}}$

Table 6. **Softmax Temperature**: large temperature in early training epochs is important. Temperature annealing refers to reducing τ from 30 to 1 linearly in the first 10 epochs. The numbers in brackets denote the performance improvement over the baseline.

Network	Top-1	Top-5
MobileNetV3-Small	67.4	86.4
MobileNetV3-Small w/o SE	$65.4_{(-2.0)}$	$85.2_{(-1.2)}$
DY-MobileNetV3-Small	$70.3_{(+2.9)}$	$88.7_{(+2.3)}$
Dy-MobileNetV3-Small w/o SE	$69.6_{(+2.2)}$	$88.4_{(+2.0)}$

Table 7. **Dynamic convolution vs Squeeze-and-Excitation (SE [13])** on MobileNetV3-Small. The numbers in brackets denote the performance improvement over the baseline. Compared with static convolution with SE, dynamic convolution *without* SE gains 2.2% top-1 accuracy.

Softmax Temperature: the temperature τ in softmax controls the sparsity of attention weights. It is important for training DY-CNNs effectively. Table 6 shows the classification accuracy for using different temperatures. $\tau = 30$ has the best performance. Furthermore, temperature annealing (reducing τ from 30 to 1 linearly in the first 10 epochs) provides additional improvement on top-1 accuracy (from 69.4% to 69.9%). Therefore, using large temperature in the early stage of training is important.

Dynamic Convolution vs Squeeze-and-Excitation (SE) [13]: MobileNetV3-Small [11] is used, in which the locations of SE layers are considered optimal as they are found by network architecture search (NAS). The results are shown in Table 7. Without using SE, the top-1 accuracy for MobileNetV3-Small drops 2%. However, DY-MobileNetV3-Small *without* SE outperforms MobileNetV3-Small with SE by 2.2% in top-1 accuracy. Combining dynamic convolution and SE gains additional 0.7% improvement. This suggests that attention over kernels and attention over output channels can work together.

5.4. Main Results

Table 8 shows the comparison between dynamic convolution and its static counterpart in three CNN architectures (MobileNetV2, MobileNetV3 and ResNet). $K = 4$ kernels are used in each dynamic convolution layer and temperature annealing is used in the training. Although we focus on efficient CNNs, we evaluate dynamic convolution on two shallow ResNets (ResNet-10 and ResNet-18) to show its effectiveness on 3×3 convolution, which is only used for the first layer in MobileNet V2 and V3. Without bells and whistles, dynamic convolution outperforms its static counterpart by a clear margin for all three architectures, with small extra computational cost ($\sim$ 4%). DY-ResNet and DY-MobileNetV2 gains more than 2.3% and 3.2% top-1 accuracy, respectively. DY-MobileNetV3-Small is 2.9% more accurate than the state-of-the-art MobileNetV3-Small.

Network	#Param	MAdds	Top-1	Top-5
MobileNetV2 $\times 1.0$	3.5M	300.0M	72.0	91.0
DY-MobileNetV2 $\times 1.0$	11.1M	312.9M	$75.2_{(3.2)}$	$92.1_{(1.1)}$
MobileNetV2 $\times 0.75$	2.6M	209.0M	69.8	89.6
DY-MobileNetV2 $\times 0.75$	7.0M	217.5M	$73.7_{(3.9)}$	$91.3_{(1.7)}$
MobileNetV2 $\times 0.5$	2.0M	97.0M	65.4	86.4
DY-MobileNetV2 $\times 0.5$	4.0M	101.4M	$69.9_{(4.5)}$	$89.0_{(2.6)}$
MobileNetV2 $\times 0.35$	1.7M	59.2M	60.3	82.9
DY-MobileNetV2 $\times 0.35$	2.8M	62.0M	$65.9_{(5.6)}$	$86.4_{(3.5)}$
MobileNetV3-Small	2.9M	66.0M	67.4	86.4
DY-MobileNetV3-Small	4.8M	68.5M	$70.3_{(2.9)}$	$88.7_{(2.3)}$
ResNet-18	11.1M	1.81G	70.4	89.7
DY-ResNet-18	42.7M	1.85G	$72.7_{(2.3)}$	$90.7_{(1.0)}$
ResNet-10	5.2M	0.89G	63.5	85.0
DY-ResNet-10	18.6M	0.91G	$67.7_{(4.2)}$	$87.6_{(2.6)}$

Table 8. ImageNet [4] classification results of DY-CNNs. The numbers in brackets denote the performance improvement over the baseline.

Input	Operator	exp size	$\#out$	n
$16 \times 12 \times B_{out}$	bneck, 5×5	768	256	2
$32 \times 24 \times 256$	bneck, 5×5	768	128	1
$64 \times 48 \times 128$	bneck, 5×5	384	128	1

Table 9. Light-weight head structures for keypoint detection. We use MobileNetV2's bottleneck residual block [27] (denoted as bneck). Each row is corresponding to a stage, which starts with a bilinear upsampling operator to scale up the feature map by 2. $\#out$ denotes the number of output channels, and n denotes the number of bottleneck residual blocks.

6. DY-CNNs for Human Pose Estimation

We use COCO 2017 dataset [19] to evaluate dynamic convolution on single-person keypoint detection. Our models are trained on `train2017`, including $57K$ images and $150K$ person instances labeled with 17 key-points. We evaluate our method on `val2017` containing 5000 images and use the mean average precision (AP) over 10 object key point similarity (OKS) thresholds as the metric.

Implementation Details: We implement two types of networks to evaluate dynamic convolution. *Type-A* follows SimpleBaseline [35] by using deconvolution in head. We use MobileNetV2 and V3 as a drop-in replacement for the backbone feature extractor and compare static convolution and dynamic convolution in the *backbone alone*. *Type-B* still uses MobileNetV2 and V3 as backbone. But it uses upsampling and MobileNetV2's bottleneck residual block in head. We compare dynamic convolution with its static counterpart in *both backbone and head*. The details of head

Type	Backbone			Head			AP	$AP^{0.5}$	$AP^{0.75}$	AP^M	AP^L	AR
	Networks	#Param	MAdds	Operator	#Param	MAdds						
A	ResNet-18	10.6M	1.77G	dconv	8.4M	5.4G	67.0	87.9	74.8	63.6	73.5	73.1
	DY-ResNet-18	42.2M	1.81G	dconv	8.4M	5.4G	$68.6_{(1.6)}$	88.4	76.1	65.3	75.1	74.6
A	MobileNetV2 $\times 1.0$	2.2M	292.6M	dconv	8.4M	5.4G	64.7	87.2	72.6	61.3	71.0	71.0
	DY-MobileNetV2 $\times 1.0$	9.8M	305.3M	dconv	8.4M	5.4G	$67.6_{(2.9)}$	88.1	75.5	64.4	74.1	73.8
A	MobileNetV2 $\times 0.5$	0.7M	93.7M	dconv	8.4M	5.4G	57.0	83.7	63.1	53.9	63.1	63.7
	DY-MobileNetV2 $\times 0.5$	2.7M	98.0M	dconv	8.4M	5.4G	$61.9_{(4.9)}$	85.8	69.7	58.9	67.9	68.4
A	MobileNetV3-Small	1.1M	62.7M	dconv	8.4M	5.4G	57.1	83.7	63.8	54.9	62.3	64.1
	DY-MobileNetV3-Small	2.8M	65.1M	dconv	8.4M	5.4G	$59.3_{(2.2)}$	84.7	66.7	56.9	64.7	66.1
B	MobileNetV2 $\times 1.0$	2.2M	292.6M	bneck	1.2M	701.1M	64.6	87.0	72.4	61.3	71.0	71.0
	DY-MobileNetV2 $\times 1.0$	9.8M	305.3M	bneck	6.3M	709.4M	$68.2_{(3.6)}$	88.4	76.0	65.0	74.7	74.2
B	MobileNetV2 $\times 0.5$	0.7M	93.7M	bneck	1.2M	701.1M	59.2	84.3	66.4	56.2	65.0	65.6
	DY-MobileNetV2 $\times 0.5$	2.7M	98.0M	bneck	6.3M	709.4M	$62.8_{(3.6)}$	86.1	70.4	59.9	68.6	69.1
B	MobileNetV3-Small	1.1M	62.7M	bneck	1.0M	664.2M	57.1	83.8	63.7	55.0	62.2	64.1
	DY-MobileNetV3-Small	2.8M	65.1M	bneck	4.9M	671.1M	$60.0_{(2.9)}$	85.0	67.8	57.6	65.4	66.7

Table 10. Keypoint detection results on COCO validation set. All models are trained from scratch. The top half uses dynamic convolution in the backbone and uses deconvolution in the head (Type A). The bottom half use MobileNetV2's bottleneck residual blocks in the head and use dynamic convolution in both the backbone and the head (Type B). Each dynamic convolution layer includes $K = 4$ kernels. The numbers in brackets denote the performance improvement over the baseline.

structure are shown in Table 9. For both types, we use $K = 4$ kernels in each dynamic convolution layer.

Training setup: We follow the training setup in [28]. The human detection boxes are cropped from the image and resized to 256×192. The data augmentation includes random rotation ($[-45°, 45°]$), random scale ($[0.65, 1.35]$), flipping, and half body data augmentation. All models are trained from scratch for 210 epochs, using Adam optimizer [17]. The initial learning rate is set as 1e-3 and is dropped to 1e-4 and 1e-5 at the 170^{th} and 200^{th} epoch, respectively. The temperature of softmax in DY-CNNs is set as $\tau = 30$.

Testing: We follow [35, 28] to use two-stage top-down paradigm: detecting person instances using a person detector and then predicting keypoints. We use the same person detectors provided by [35]. The keypoints are predicted on the average heatmap of the original and flipped images by adjusting the highest heat value location with a quarter offset from the highest response to the second highest response.

Main Results and Ablations: Firstly we compare dynamic convolution with its static counterpart in the backbone (*Type-A*). The results are shown in the top half of Table 10. Dynamic convolution gains 1.6, 2.9, 2.2 AP for ResNet-18, MobileNetV2 and MobileNetV3-Small, respectively.

Secondly, we replace the heavy deconvolution head with light-weight upsampling and MobileNetV2's bottleneck residual blocks (*Type-B*) to make the whole network small and efficient. Thus, we can compare dynamic convolution with its static counterpart in both backbone and head. The results are shown in the bottom half of Table 10. Similar to *Type-A*, dynamic convolution outperforms its static counterpart by a clear margin. It gains 3.6 and 2.9 AP for MobileNetV2 and MobileNetV3-Small, respectively.

Backbone	Head	AP	$AP^{0.5}$	$AP^{0.75}$
static	static	59.2	84.3	66.4
static	dynamic	$60.3_{(1.1)}$	84.9	67.3
dynamic	static	$62.3_{(3.1)}$	85.6	70.0
dynamic	dynamic	$62.8_{(3.6)}$	86.1	70.4

Table 11. Keypoint detection results of using dynamic convolution in backbone and head separately. We use MobileNetV2 $\times 0.5$ as backbone and use the light-weight head structure discussed in Table 9. The numbers in brackets denote the performance improvement over the baseline. Dynamic convolution can improve AP at both the backbone and the head.

We perform an ablation to investigate the effects of dynamic convolution at backbone and head separately (Table 11). Even though most of improvement comes from the dynamic convolution at the backbone, dynamic convolution at the head is also helpful. This is mainly because the backbone has more convolution layers than the head.

7. Conclusion

In this paper, we introduce dynamic convolution, which aggregates multiple convolution kernels dynamically based upon their attentions for each input. Compared to its static counterpart (single convolution kernel per layer), it significantly improves the representation capability with negligible extra computation cost, thus is more friendly to efficient CNNs. Our dynamic convolution can be easily integrated into existing CNN architectures. By simply replacing each convolution kernel in MobileNet (V2 and V3) with dynamic convolution, we achieve solid improvement for both image classification and human pose estimation. We hope dynamic convolution becomes a useful component for efficient network architectures.

A. Appendix

In this appendix, we report running time and perform additional analysis for our dynamic convolution method.

A.1. Inference Running Time

We report the running time of dynamic MobileNetV2 (DY-MobileNetV2) with four different width multipliers ($\times 1.0$, $\times 0.75$, $\times 0.5$, and $\times 0.35$) and compare with its static counterpart (MobileNetV2 [27]) in Table 12. We use a single-threaded core of Intel Xeon CPU E5-2650 v3 (2.30GHz) to measure running time (in milliseconds). The running time is calculated by averaging the inference time of 5,000 images with batch size 1. Both MobileNetV2 and DY-MobileNetV2 are implemented using PyTorch [25].

Compared with its static counterpart, DY-MobileNetV2 consumes about 10% more running time and 4% more Multi-Adds. The overhead of running time is higher than Multi-Adds. We believe this is because the optimizations of global average pooling and small inner-product operations are not as efficient as convolution. With the small additional computational cost, our dynamic convolution method significantly improves the model performance.

A.2. Dynamic Convolution in Shallower and Thinner Networks

Figure 7 shows that the shallower DY-MobileNetV2 (depth $\times 0.5$) has better trade-off between accuracy and computational cost than the deeper MobileNetV2 (depth $\times 1.0$), even though shallower networks (depth $\times 0.5$) have performance degradation for both DY-MobileNetV2 and MobileNetV2. Improvement on shallow networks is useful as they are friendly to parallel computation. Furthermore, dynamic convolution achieves more improvement for thinner and shallower networks with small width/depth multipliers. This is because thinner and shallower networks are underfitted due to their limited model size and dynamic convolution significantly improves their capability.

Network	Top-1	MAdds	CPU (ms)
MobileNetV2 $\times 1.0$	72.0	300.0M	127.9
DY-MobileNetV2 $\times 1.0$	$75.2_{(3.2)}$	312.9M	141.2
MobileNetV2 $\times 0.75$	69.8	209.0M	99.5
DY-MobileNetV2 $\times 0.75$	$73.7_{(3.9)}$	217.5M	110.5
MobileNetV2 $\times 0.5$	65.4	97.0M	69.6
DY-MobileNetV2 $\times 0.5$	$69.9_{(4.5)}$	101.4M	77.4
MobileNetV2 $\times 0.35$	60.3	59.2M	61.1
DY-MobileNetV2 $\times 0.35$	$65.9_{(5.6)}$	62.0M	67.4

Table 12. **Inference running time** of DY-MobileNetV2 [27] on ImageNet [4] classification. We use dynamic convolution with $K = 4$ kernels for all convolution layers in DY-MobileNetV2 except the first layer. CPU: CPU time in milliseconds measured on a single core of Intel Xeon CPU E5-2650 v3 (2.30GHz). The running time is calculated by averaging the inference time of 5,000 images with batch size 1. The numbers in brackets denote the performance improvement over the baseline.

A.3. Example: Learning XOR

To make the idea of dynamic perceptron more concrete, we use it on a simple task, i.e. learning the XOR function. In this example, we want our network to perform correctly on the four points $\mathbb{X} = \{[0,0]^T, [0,1]^T, [1,0]^T, [1,1]^T\}$. Compared with the solution using *two* static perceptron layers [6] as follows:

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}\} \\ \boldsymbol{w} &= \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \boldsymbol{b} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \end{aligned} \tag{4}$$

dynamic perception only needs a *single* layer as follows:

$$\begin{aligned} \boldsymbol{y} &= \sum_{k=1}^{2} \left[\left(\pi_k(\boldsymbol{x}) \tilde{\boldsymbol{W}}_k^T \right) \boldsymbol{x} + \pi_k(\boldsymbol{x}) \tilde{\boldsymbol{b}}_k \right] \\ \tilde{\boldsymbol{W}}_1 &= \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix}, \tilde{\boldsymbol{b}}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tilde{\boldsymbol{W}}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \tilde{\boldsymbol{b}}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \end{aligned} \tag{5}$$

where the attentions are $\pi_1(\boldsymbol{x}) = x_2, \pi_2(\boldsymbol{x}) = 1 - x_2$. This example demonstrates that dynamic perceptron has more representation power due to the non-linearity.

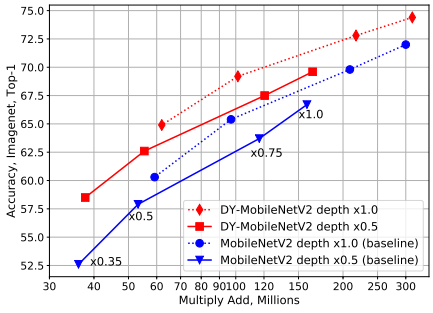

Figure 7. **Shallower DY-MobileNetV2 vs Deeper MobileNetV2**. The shallower DY-MobileNetV2 (depth $\times 0.5$) has better trade-off between accuracy and computational cost than the deeper MobileNetV2 (depth $\times 1.0$). To make comparison fair, we also plot the deeper DY-MobileNetV2 and shallower MobileNetV2. For both DY-MobileNetV2 and MobileNetV2, deeper networks have better performance. Best viewed in color.

References

[1] Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. *ArXiv*, abs/1908.09791, 2019.

[2] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3123–3131. Curran Associates, Inc., 2015.

[4] 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*, pages 248–255. Ieee, 2009.

[5] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[7] 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, 2019.

[8] Song Han, Huizi Mao, and William Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 10 2016.

[9] 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*, pages 770–778, 2016.

[10] 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 *The European Conference on Computer Vision (ECCV)*, September 2018.

[11] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019.

[12] 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.

[13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[14] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*, 2018.

[15] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[16] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *The IEEE International Conference on Computer Vision (ICCV)*, 2009.

[17] Diederick P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[18] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, pages 2181–2191. 2017.

[19] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[21] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[22] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[23] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[24] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[25] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[26] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[27] 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*, pages 4510–4520, 2018.

[28] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *CVPR*, 2019.

[29] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[30] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[31] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[32] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[33] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. 2017.

[34] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[35] Bin Xiao, Haiping Wu, and Yichen Wei. Simple baselines for human pose estimation and tracking. In *European conference on computer vision*, 04 2018.

[36] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019.

[37] Brandon Yang, Gabriel Bender, Quoc V. Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. In *NeurIPS*, 2019.

[38] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. Quantization networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[39] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2019.

[40] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[41] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[42] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[43] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. In *International Conference on Learning Representations (ICLR)*, 2017.

[44] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2017.

[45] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.