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.

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 $\{\tilde{W}_k, \tilde{b}_k\}$ instead of using a single convolution kernel per layer (see Figure 2). These convolution kernels are aggregated dynamically $W = \sum_k \pi_k(x) \tilde{W}_k$ for each individual input $x$ (e.g. im-
age) via input dependent attention $\pi_k(x)$. The biases are aggregated using the same attention $b = \sum_k \pi_k(x) \hat{b}_k$. 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 $\{\pi_k(x)\}$ 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 $\sum_k \pi_k(x) = 1$ to facilitate the learning of attention model $\pi_k(x)$, and (b) flattening attention (near-uniform) in early training epochs to facilitate the learning of convolution kernels $\{W_k, \hat{b}_k\}$. 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 MobileNetV2 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, 13, 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, 18, 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, 43, 39, 14] that focus on skipping part of an existing model based on input image. D2NN [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 [11] 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 [34], 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, 56] 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 $y = g(W^T x + b)$, where $W$ and $b$ 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 $\{\tilde{W}_k^T x + \tilde{b}_k\}$ as follows:

$$
\begin{align*}
    y &= g(\tilde{W}^T x + \tilde{b}) \\
    \tilde{W}(x) &= \sum_{k=1}^{K} \pi_k(x) \tilde{W}_k, \tilde{b}(x) = \sum_{k=1}^{K} \pi_k(x) \tilde{b}_k \\
    \text{s.t. } 0 \leq \pi_k(x) \leq 1, \sum_{k=1}^{K} \pi_k(x) = 1,
\end{align*}
$$

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

**Attention:** the attention weights $\{\pi_k(x)\}$ are not fixed, but vary for each input $x$. They represent the optimal aggregation of linear models $\{\tilde{W}_k^T x + \tilde{b}_k\}$ for a given input. The aggregated model $\tilde{W}^T(x)x + \tilde{b}(x)$ 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 $\{\pi_k(x)\}$, and (b) aggregating parameters based upon attention $\sum_k \pi_k \tilde{W}_k$ and $\sum_k \pi_k \tilde{b}_k$. The additional computational cost should be significantly less than the cost of computing $W^T x + b$. Mathematically, the computational constraint can be represented as follows:

$$
O(\tilde{W}^T x + \tilde{b}) \gg O \left( \sum_k \pi_k \tilde{W}_k \right) + O \left( \sum_k \pi_k \tilde{b}_k \right) + O(\pi(x))
$$

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.

### 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 $\{\pi_k(x)\}$ (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 $O(\pi(x)) = HWC_{in} + C_{in}^2/4 + C_{in}K/4$ Multi-Adds. This is much less than the computational cost of convolution, i.e. $O(\tilde{W}^T x + \tilde{b}) = HWC_{in}C_{out}D_k^2$ Multi-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 negligible 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...
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., $1 \times 1$ conv, $3 \times 3$ conv, group convolution, depthwise 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 $\tilde{W}_k$ to denote a convolution kernel and ignore bias $\delta_k$, 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 $\{\tilde{W}_k\}$ and attention model $\pi_k(x)$ 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 $\pi_k(x)$. Specifically, we have the constraint $\sum_k \pi_k(x) = 1$ in Eq. (1) to keep the aggregated kernel $\tilde{W} = \sum_k \pi_k \tilde{W}_k$ within the convex hull of $\{\tilde{W}_k\}$ in the kernel space. Figure 4 shows an example with 3 convolution kernels. The constraint $0 \leq \pi_k(x) \leq 1$ only keeps the aggregated kernel within the two pyramids. The sum-to-one constraint further compresses the kernel space to a triangle. A red line is compressed into a dot by normalizing attention sum. Best viewed in color.

#### 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 $\{\tilde{W}_k\}$ during early training epochs. This is because near-uniform attention enables more convolution kernels to be optimized simultaneously.

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 $1 \times 1$ 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:

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

where $z_k$ is the output of the second FC layer in attention branch (see Figure 7) and $\tau$ is the temperature. The original softmax is a special case ($\tau = 1$). As $\tau$ 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 $\tau$ 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
Table 2. Comparison between DY-CNNs and the concurrent work (DynConv [37]) on ImageNet classification using MobileNetV2 × 1.0 and × 0.5.

<table>
<thead>
<tr>
<th>×1.0</th>
<th>Method</th>
<th>#Kernels</th>
<th>#Param</th>
<th>MAdds</th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CondConv [37]</td>
<td>8</td>
<td>27.5M</td>
<td>392M</td>
<td>74.6</td>
<td>68.8</td>
</tr>
<tr>
<td></td>
<td>DY-CNNs (ours)</td>
<td>4</td>
<td>11.1M</td>
<td>312.9M</td>
<td>75.2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>×0.5</th>
<th>Method</th>
<th>#Kernels</th>
<th>#Param</th>
<th>MAdds</th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CondConv [37]</td>
<td>8</td>
<td>15.5M</td>
<td>41M</td>
<td>68.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DY-CNNs (ours)</td>
<td>4</td>
<td>4M</td>
<td>101.4M</td>
<td>69.9</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Inspecting DY-CNN using different kernel aggregations. DY-MobileNetV2 × 0.5 is used. The proper aggregation of convolution kernels \( \{ W_k \} \) using attention \( \pi_k(x) \) 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 \( x \) for another image \( x' \). The poor performance for the bottom four aggregations validates that the DY-CNN is dynamic.

<table>
<thead>
<tr>
<th>Kernel Aggregation</th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>attention: ( \sum \pi_k(x)W_k )</td>
<td>69.4</td>
<td>88.6</td>
</tr>
<tr>
<td>average: ( \frac{1}{K} \sum W_k )</td>
<td>36.0</td>
<td>61.5</td>
</tr>
<tr>
<td>max: ( \tilde{W}_{\arg\max_k(\pi_k)} )</td>
<td>0.1</td>
<td>0.5</td>
</tr>
<tr>
<td>shuffle per image: ( \sum \pi_j(x)\tilde{W}_{k,j} ) ( j \neq k )</td>
<td>14.8</td>
<td>30.5</td>
</tr>
<tr>
<td>shuffle across images: ( \sum \pi_k(x)\tilde{W}_{k} ) ( x' )</td>
<td>27.3</td>
<td>48.4</td>
</tr>
</tbody>
</table>

Figure 5. Training and validation errors for using different soft-max 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.

epochs, it has significantly larger kernel space (two pyramids in Figure 3) than our method (the shaded triangle in Figure 4). Thus, learning attention model \( \pi_k(x) \) 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 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 (×1.0 and ×0.75). The dropout rate is 0.2 and 0.1 for ×1.0 and ×0.75, respectively. Mixup [41] is used for ×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.

5.2. Inspecting DY-CNN

We inspect if DY-CNN is dynamic, using DY-MobileNetV2 × 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 \( \sum \tilde{W}_k/K \), choosing the convolution kernel with the maximum attention \( \tilde{W}_{\arg\max_k(\pi_k)} \), and random shuffling attention over kernels per image \( \sum \pi_j(x)\tilde{W}_{k,j} \) \( j \neq k \). 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...
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 142 and 72), 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, 562 and 282), 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 hyperparameter $K$ controls the model complexity. Figure 9 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 ×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.
Softmax Temperature: the temperature $\tau$ 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 $\tau$ 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 $3 \times 3$ convolution, which is only used for the first layer in MobileNetV2 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.

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 keypoints. 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
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.

<table>
<thead>
<tr>
<th>Type</th>
<th>Backbone Networks</th>
<th>#Param</th>
<th>MAdds</th>
<th>Operator</th>
<th>Head</th>
<th>#Param</th>
<th>MAdds</th>
<th>AP</th>
<th>AP(^0.5)</th>
<th>AP(^0.75)</th>
<th>AP(^M)</th>
<th>AP(^L)</th>
<th>AR</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ResNet-18</td>
<td>10.6M</td>
<td>1.77G</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>67.0</td>
<td>87.9</td>
<td>74.8</td>
<td>63.6</td>
<td>73.5</td>
<td>73.1</td>
</tr>
<tr>
<td></td>
<td>DY-ResNet-18</td>
<td>42.2M</td>
<td>1.81G</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>68.0(^{1.6})</td>
<td>88.4</td>
<td>76.1</td>
<td>65.3</td>
<td>75.1</td>
<td>74.6</td>
</tr>
<tr>
<td>A</td>
<td>MobileNetV2 (\times 1.0)</td>
<td>2.2M</td>
<td>292.6M</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>64.7</td>
<td>87.2</td>
<td>72.6</td>
<td>61.3</td>
<td>71.0</td>
<td>71.0</td>
</tr>
<tr>
<td></td>
<td>DY-MobileNetV2 (\times 1.0)</td>
<td>9.8M</td>
<td>305.3M</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>67.0(^{2.9})</td>
<td>88.1</td>
<td>75.5</td>
<td>64.4</td>
<td>74.1</td>
<td>73.8</td>
</tr>
<tr>
<td>A</td>
<td>MobileNetV2 (\times 0.5)</td>
<td>0.7M</td>
<td>93.7M</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>62.8</td>
<td>83.7</td>
<td>63.1</td>
<td>53.9</td>
<td>63.1</td>
<td>63.7</td>
</tr>
<tr>
<td></td>
<td>DY-MobileNetV2 (\times 0.5)</td>
<td>2.7M</td>
<td>98.0M</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>61.9(^{4.9})</td>
<td>85.8</td>
<td>69.7</td>
<td>58.9</td>
<td>67.9</td>
<td>68.4</td>
</tr>
<tr>
<td>A</td>
<td>MobileNetV3-Small</td>
<td>1.1M</td>
<td>62.7M</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>57.1</td>
<td>83.7</td>
<td>63.8</td>
<td>54.9</td>
<td>62.3</td>
<td>64.1</td>
</tr>
<tr>
<td></td>
<td>DY-MobileNetV3-Small</td>
<td>2.8M</td>
<td>65.1M</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>dconv 8.4M</td>
<td>5.4G</td>
<td>59.3(^{2.2})</td>
<td>84.7</td>
<td>66.7</td>
<td>56.9</td>
<td>64.7</td>
<td>66.1</td>
</tr>
<tr>
<td>B</td>
<td>MobileNetV2 (\times 1.0)</td>
<td>2.2M</td>
<td>292.6M</td>
<td>bneck 1.2M</td>
<td>701.1M</td>
<td>bneck 1.2M</td>
<td>701.1M</td>
<td>64.6</td>
<td>87.0</td>
<td>72.4</td>
<td>61.3</td>
<td>71.0</td>
<td>71.0</td>
</tr>
<tr>
<td></td>
<td>DY-MobileNetV2 (\times 1.0)</td>
<td>9.8M</td>
<td>305.3M</td>
<td>bneck 1.2M</td>
<td>709.4M</td>
<td>bneck 1.2M</td>
<td>709.4M</td>
<td>68.2(^{3.6})</td>
<td>88.4</td>
<td>76.0</td>
<td>65.0</td>
<td>74.7</td>
<td>74.2</td>
</tr>
<tr>
<td>B</td>
<td>MobileNetV2 (\times 0.5)</td>
<td>0.7M</td>
<td>93.7M</td>
<td>bneck 1.2M</td>
<td>701.1M</td>
<td>bneck 1.2M</td>
<td>701.1M</td>
<td>59.2</td>
<td>84.3</td>
<td>66.4</td>
<td>56.2</td>
<td>65.0</td>
<td>65.6</td>
</tr>
<tr>
<td></td>
<td>DY-MobileNetV2 (\times 0.5)</td>
<td>2.7M</td>
<td>98.0M</td>
<td>bneck 1.2M</td>
<td>709.4M</td>
<td>bneck 1.2M</td>
<td>709.4M</td>
<td>62.8(^{3.6})</td>
<td>86.1</td>
<td>70.4</td>
<td>59.9</td>
<td>68.6</td>
<td>69.1</td>
</tr>
<tr>
<td>B</td>
<td>MobileNetV3-Small</td>
<td>1.1M</td>
<td>62.7M</td>
<td>bneck 1.0M</td>
<td>664.2M</td>
<td>bneck 1.0M</td>
<td>664.2M</td>
<td>57.1</td>
<td>83.8</td>
<td>63.7</td>
<td>55.0</td>
<td>62.2</td>
<td>64.1</td>
</tr>
<tr>
<td></td>
<td>DY-MobileNetV3-Small</td>
<td>2.8M</td>
<td>65.1M</td>
<td>bneck 4.9M</td>
<td>671.1M</td>
<td>bneck 4.9M</td>
<td>671.1M</td>
<td>60.0(^{2.9})</td>
<td>85.0</td>
<td>67.8</td>
<td>57.6</td>
<td>65.4</td>
<td>66.7</td>
</tr>
</tbody>
</table>

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 (×1.0, ×0.75, ×0.5, and ×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 ×0.5) has better trade-off between accuracy and computational cost than the deeper MobileNetV2 (depth ×1.0), even though shallower networks (depth ×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.

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 \(X = \{[0, 0]^T, [0, 1]^T, [1, 0]^T, [1, 1]^T\}\). Compared with the solution using two static perceptron layers \[6\] as follows:

\[
y = w^T \max\{0, W^T x + b\}
\]

\[
w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, b = \begin{bmatrix} 0 \\ -1 \end{bmatrix},
\]

(4)

dynamic perception only needs a single layer as follows:

\[
y = \sum_{k=1}^{2} \left( \pi_k(x) \tilde{W}_k^T x + \pi_k(x) \tilde{b}_k \right)
\]

\[
\tilde{W}_1 = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}, \tilde{b}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \tilde{W}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \tilde{b}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},
\]

(5)

where the attentions are \(\pi_1(x) = x_2, \pi_2(x) = 1 - x_2\). 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 ×0.5) has better trade-off between accuracy and computational cost than the deeper MobileNetV2 (depth ×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.](image-url)

<table>
<thead>
<tr>
<th>Network</th>
<th>Top-1</th>
<th>MAdds</th>
<th>CPU (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNetV2 ×1.0</td>
<td>72.0</td>
<td>300.0M</td>
<td>127.9</td>
</tr>
<tr>
<td>DY-MobileNetV2 ×1.0</td>
<td>75.2 (3.2)</td>
<td>312.9M</td>
<td>141.2</td>
</tr>
<tr>
<td>MobileNetV2 ×0.75</td>
<td>69.8</td>
<td>209.0M</td>
<td>99.5</td>
</tr>
<tr>
<td>DY-MobileNetV2 ×0.75</td>
<td>73.7 (3.9)</td>
<td>217.5M</td>
<td>110.5</td>
</tr>
<tr>
<td>MobileNetV2 ×0.5</td>
<td>69.4</td>
<td>97.0M</td>
<td>69.6</td>
</tr>
<tr>
<td>DY-MobileNetV2 ×0.5</td>
<td>69.9 (4.5)</td>
<td>101.4M</td>
<td>77.4</td>
</tr>
<tr>
<td>MobileNetV2 ×0.35</td>
<td>60.3</td>
<td>59.2M</td>
<td>61.1</td>
</tr>
<tr>
<td>DY-MobileNetV2 ×0.35</td>
<td>65.9 (5.6)</td>
<td>62.0M</td>
<td>67.4</td>
</tr>
</tbody>
</table>

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.
References


