Hybrid Composition with IdleBlock: More Efficient Networks for Image Recognition

Bing Xu, Andrew Tulloch, Yunpeng Chen[*], Xiaomeng Yang, Lin Qiao
Facebook AI

Abstract

We propose a new building block, IdleBlock, which naturally prunes connections within the block. To fully utilize the IdleBlock we break the tradition of monotonic design in state-of-the-art networks, and introducing hybrid composition with IdleBlock. We study hybrid composition on MobileNet v3 [9] and EfficientNet-B0 [24], two of the most efficient networks. Without any neural architecture search, the deeper MobileNet v3 with hybrid composition design surpasses possibly all state-of-the-art image recognition network designed by human experts or neural architecture search algorithms. Similarly, the hybridized EfficientNet-B0 networks are more efficient than previous state-of-the-art networks with similar computation budgets. These results suggest a new simpler and more efficient direction for network design and neural architecture search.

1. Introduction

Convolutional Neural Networks (CNN) have dominated computer vision tasks in recent years. Since AlexNet [14], the computer vision community has sought improved CNN designs to make the backbone network more powerful and more efficient. Remarkable single branch backbones include Network in Network [16], VGGNet [22], ResNet [8], DenseNet [12], ResNext [27], MobileNet v1/v2/v3 [10, 21, 9], and ShuffleNet v1/v2 [30, 19]. In recent years, backbone networks with multiple resolutions have also attracted attention from the research community. To learn with multiple resolutions, researchers design complex connections inside a block to handle information exchanges of different resolutions. Some efficient examples of this approach include the MultiGrid-Conv [13], OctaveConv [5], and HRNet [23]. All these contributions have significantly developed the philosophy of backbone network design.

To design more efficient CNNs, there are two mainstream efforts: Neural Architecture Search (NAS) and Network Pruning (NP). The motivation of neural architecture search is: given a constraint on computation resources, NAS attempts to automatically determine the best network connections, block designs and hyper-parameters. Hyperparameter searching is a classic topic in machine learning, and in our paper we refer NAS as specially to searching over the connections and block design of neural network. The motivation of network pruning is: given a pretrained network, an automatic algorithm is able to remove unimportant connections, to reduce computation and parameters.

[*] Work was done during internship at Facebook

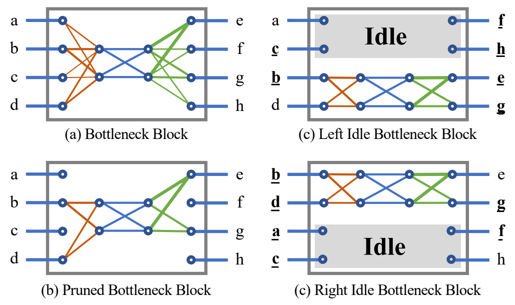
a
b
c
d
e
f
g
h
(a) Bottleneck Block
Idle
(c) Left Idle Bottleneck Block
(b) Pruned Bottleneck Block
(c) Right Idle Bottleneck Block

Figure 1: The motivation of our proposed Idle design. In Idle design information exchange is applied outside of the Idled blocks.

In contrast to NAS over connections and NP, EfficientNet [24] provides a joint hyperparameter for a backbone: the depth scaling factor d, width scaling factor w and input resolution scaling factor r, which is referred as compound scaling factors. Based on a variant of MobileNet v3, these jointly-searched scaling factors make the EfficientNet family 5× to 10× more efficient than all previous backbones in measured of computation cost (MAdds) or the number of parameters.

In this work, We revisit the modern workflow of obtaining efficient convolutional networks, and partition the problem into two separate steps. In the first step, we design a network architecture, and in the second step, we prune connections from the network.

In the first step, we note a common pattern in both human expert-designed architectures and searched architectures: for each backbone, the architecture is charac-

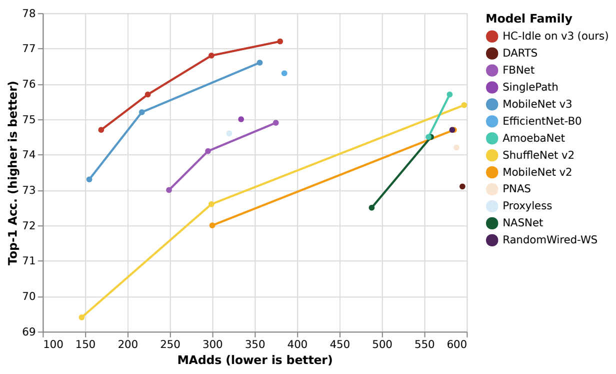

Figure 2: ImageNet Top-1 Accuracy / Computation cost trade-off for small model (MAdds < 600M). In this figure we compare Hybrid Composition with IdleBlock on MobileNet v3 with state-of-art human expert-designed networks and NAS networks.

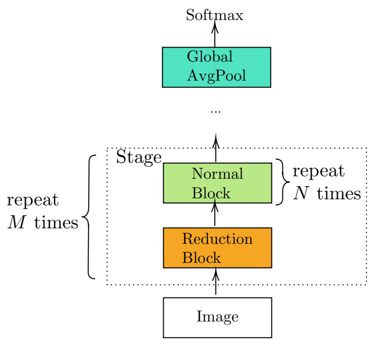

Figure 3: Monotonous design.

terised by the design of a normal block and a reduction block. At the beginning of each stage, we insert a reduction block, and repeatedly stack the normal block. We repeat each stage multiple times, and for each stage we may have different number of normal blocks. We call this design pattern Monotonous Design (Figure 3). For example, ResNet monotonically repeats a Bottleneck block, ShuffleNet monotonically repeats a ShuffleBlock, MobileNet v2/v3 and EfficientNet monotonically repeats and Inverted Residual Block (MBBlock), NASNet repeats a Normal Cell, and FBNet repeats a variant of MBBlock with different hyper-parameters. For all the state-of-art networks, to our best knowledge, the blocks are guaranteed to have full information exchange. In the second step, some connections are pruned, and it is not be guaranteed that each block has full information exchange.

In this paper, we seek to design a more efficient network for image recognition tasks, by considering pruning in the network design step. We create a new block design methodology: *Idle*. In the Idle design, a subspace of the input is not transformed: it is simply idle and passed directly to

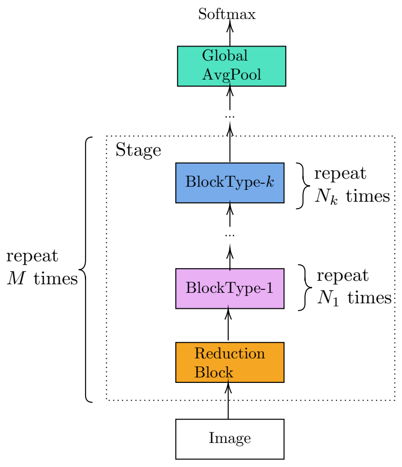

Figure 4: Hybrid Composition

the output (Figure 2). We also break the monotonous design constraint in state-of-the-art architectures. We call our non-monotonous composition method *Hybrid Composition* (HC) (Figure 4).

The results are initially as expected — if we monotonically compose a network with IdleBlock, we obtain a pruned network with acceptable accuracy loss. If we use hybrid composition with IdleBlock (and MBBlock), we significantly reduce the accuracy loss while substantially saving computation. The surprising result is this: **leveraging the pruned computation budget from IdleBlock to go deeper with hybrid composition achieves new state-of-art network architectures — without complex multi-resolution design or neural architecture search.**

The paper is organized as follows. In Section 2, We discuss modern convolutional network block design, and introduce the Idle design and IdleBlock. In Section 3, we introduce Hybrid Composition (HC) with IdleBlock. In Section 4 we study applying HC with IdleBlock to MobileNet v3 and EfficientNet-B0, which leads to new state-of-the-art network architectures under equivalent computation budgets. We also conduct ablation studies on our design choice. Section 5 contains conclusion and future work.

2. Idle & IdleBlock: Motivation and Discussion

In this section, we will revisit key milestones in convolution building block design. We will then introduce our Idle design and Idled version of MBBlock: IdleBlock.

2.1. Bottleneck Block & Inverted Residual Block

The Inverted Residual Block (MBBlock) (Figure 6) is the fundamental building block behind success of MobileNet v2/v3 and EfficientNet. In MBBlock, a pointwise convolution will expand input dimension by a factor of r,

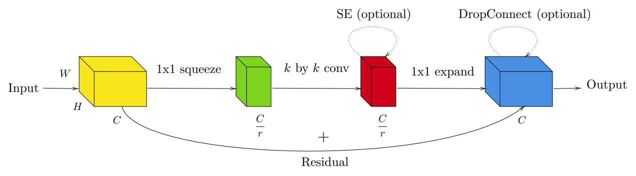

Figure 5: Bottleneck block. The Bottleneck block seeks to reduce computation of spatial convolution. Each block consists of expanded input and output, without nonlinearities. The residual connection is between expanded representations.

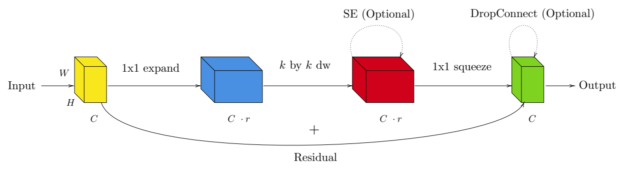

Figure 6: Inverted Residual Block (MBBlock). The Inverted Residual Block seeks to extract rich spatial information from an expanded projection. Each block consist of narrow inputs and outputs, without nonlinearities. The residual connection is between narrowed representations.

apply a depthwise convolution to the expanded features, then apply another pointwise convolution to squeeze the high dimension features into a lower dimension (usually the same as input to allow residual addition). The key difference between the Bottleneck block [8, 10] design (Figure 5 and MBBlock is — Bottleneck applies a spatial transformation ($k \times k$ convolution or $k \times k$ depthwise convolution) on a narrowed feature map, and MBBlock applies a spatial transformation on an expanded feature map. The Bottleneck design is proposed in ResNet, where 3×3 convolutions are the most computationally intensive component. With 3×3 convolutions, narrowed feature maps significantly reduce computation cost. However, when depthwise convolutions are used, the spatial transformation is no longer a computation bottleneck. A $k \times k$ depthwise convolution corresponds to running a linear regression with k^2 input elements repeatly in each channel. Intuitively, with more channels, the spatial transformation capacity will be higher. Empirically, with similar computational budgets (MAdds), MobileNet v1 in Bottleneck design achieves 70.6% Top 1 accuracy on ImageNet [6], while MobileNet v2 (1.4) achieves 74.7% Top 1 accuracy [21]. The accuracy gap between MobileNet v1 and v2 indicates that for depthwise convolutions, an expanded feature map is helpful to improve a network's expression power.

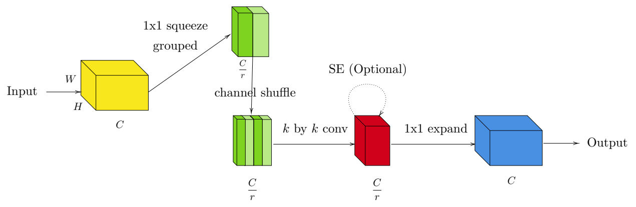

Figure 7: ShuffleBlock v1. ShuffleBlock v1 is an extension of Bottleneck block (Figure 5). To reduce computation of the narrowed representation, a grouped pointwise operation is introduced, followed by channel shuffle operation.

2.2. ShuffleBlock v1/v2

ShuffleBlock v1/v2 are depthwise convolution blocks that heavily rely on the channel shuffle operation — a lightweight information exchange operation.

ShuffleBlock v1 (Figure 7) is inherited from Bottleneck block. A grouped pointwise convolution is used to replace the first pointwise convolution in Bottleneck block to narrow the channel dimension. A channel shuffle operator is followed by grouped pointwise convolution operator to provide information exchange between groups. The block topology is the same as the Bottleneck block.

ShuffleBlock v2 (Figure 8) is a following work of ShuffleBlock v1. Similar to Bottleneck block and ShuffleBlock v1, ShuffleBlock v2s input and output are expanded feature maps. To further reduce computation cost, the group pointwise convolution which is used for narrowing feature map is removed. Instead, expanded feature map is split into two equal channel narrow feature maps: one narrowed feature map is transformed with a special Bottleneck block without internal dimension change, the other feature map keeps the same until it is shuffled with transformed feature map.

However, the channel shuffle operation is not efficiently realizable on many heterogeneous hardware, such as new neural network accelerators. For example on iPhone equipped with Apple Neural Engine (ANE) using CoreML, networks without channel shuffle operations run up to 10 times slower than equivalent networks without channel shuffles.

2.3. "Idle" Design

When we think about design principles for an efficient block structure, attempting to reduce computation (MAdds) is an obvious direction. For example, to reduce computation in a $k \times k$ spatial convolution, depthwise convolutions can be used to replace the original convolution operator (as in MBBlock).

When we optimize directly for modern hardware platforms, there are more challenges than simply reducing MAdds. For example, post-training pruning will reduce computation, but without very specialized implementations

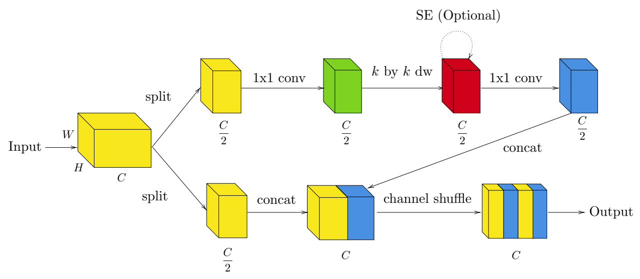

Figure 8: ShuffleBlock v2. ShuffleBlock v2 removed the grouped pointwise operation – instead, using split to obtain a narrowed representation. Similar to Bottleneck block (Figure 5) and ShuffleBlock v1 (Figure 7), each block consist of an expanded input and output.

and high levels of sparsity, it is hard to realize these computational improvements in practice. We seek to introduce a structured pruned topology into our block design, which allows us to achieve real-world speed ups. With this insight, we create a new design pattern: *Idle*, which aims to directly pass a subspace of the input to the output tensor, without any transformations. Figure 1 demonstrates motivation of Idle and network pruning. In Idle design, we introduce an idle factor $\alpha \in (0, 1)$, which is also can be viewed as pruning factor. Given an input tensor x with C channels, the tensor will be sliced to two branches: one active branch x_1 containing $C \cdot (1 - \alpha)$ channels, and outputting a tensor y_1 with $C \cdot (1 - \alpha)$ channels; and the other idle branch x_2 contains $C \cdot \alpha$ channels, copied directly to the output tensor y with C channels directly.

Contrasted with Residual Connections

With residual connections, the input tensor x is added to the output tensor. In the Idle design, $C \cdot \alpha$ channels of input tensor x will be copied into the output tensor directly, without any elementwise addition.

Contrasted with Dense Connections

In a densely connected block, the entire input x is part of the output y, and entire input x in used in the block transformation. In Idle design, only $C \cdot \alpha$ channels of x are copied into the output tensor directly, and the other $C \cdot (1 - \alpha)$ channels are used in the block transformation.

Contrasted with ShuffleBlock v2

The Idle design is closely related to ShuffleBlock v2 but with a few differences. First, ShuffleBlock v2 requires expanded feature map as input and outputs an expanded feature map; Idle is a design pattern doesnt require on input/output dimension. Second, in ShuffleBlock v2, all information will be exchanged by using channel shuffle operator; In Idle design, we intend to dont apply any change for a subspace of input. Third, Idle designs motivation is to design a pre-pruned structure, ShuffleBlock v2 aims to obtain a narrowed feature map for spatial transformation. We can view ShuffleBlock v2 in a perspective of Idled Bottleneck block with extra channel shuffle.

2.4. IdleBlock

We summarize a few intuitive and empirical lessons we learned from ShuffleBlock v1/v2 and MBBlock:

1. We need to apply the depthwise convolution on an expanded feature map. (MobileNet v1 vs MobileNet v2)

2. Grouped convolutions are not necessary. (ShuffleNet v1 vs ShuffleNet v2)

3. The channel shuffle operation is not friendly to various accelerators, and should be avoided.

Based on these lessons, we introduce IdleBlock, an Idled version of MBBlock.

There are two variants of IdleBlock. If we use the concatenation function concat(y_1, x_2) when constructing the output tensor from the two splits, we refer to the it as an L-IdleBlock (Figure 9), and if we use the concatenation function concat(x_2, y_1), we refer to it as an R-IdleBlock. If an information exchange block is strictly followed by an IdleBlock, the L-IdleBlock and R-IdleBlock are equivalent. When stacking two or more IdleBlocks, a mix of L/R-IdleBlock units is different to a monotonic composition of L/R-IdleBlock units.

A key distinction of mixed composition is the enhanced receptive field of the stacked output. Using two stacked IdleBlocks as example. Given inputs with receptive field R_0, if we monotonically stack two R-IdleBlocks, the first output branchs receptive field will remain as R_0 as the first branch is idle, while the second output branchs receptive field is $R_0 \cdot k^2$ as the second branch of input is updated by two k by k depthwise convolution operations. If we mix an L- and R-IdleBlock, L/R-IdleBlock, both of the output branches will be updated with one k by k depthwise convolution operator, which will change the receptive field of both output branches to $R_0 \cdot k$.

Given an MBBlock with input tensor shape $(1, C, H, W)$, expansion factor r, stride s, and depthwise convolution kernel k, the theoretical computation complexity is

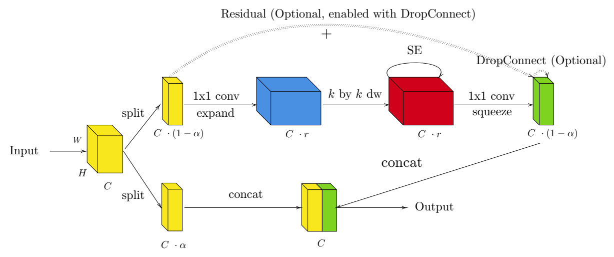

Figure 9: L-IdleBlock. IdleBlock is an idled version of MBBlock (Figure 6. Similar to MBBlock, each block consist of narrowed input and output without nonlinearities. L-IdleBlock concatenate non-transformed branch first, followed by transformed branch. R-IdleBlock concatenate in reversed way. L/R-IdleBlock won't change input/output shape or channel dimension.

$$\begin{aligned}\sum(2 \cdot r \cdot C^2 \cdot HW, \quad & \text{//1x1 expand} \\ 2 \cdot r \cdot C \cdot (k^2 \cdot \frac{HW}{s^2}), \quad & \text{//k by k depthwise} \\ 2 \cdot r \cdot C^2 \cdot \frac{HW}{s^2}) \quad & \text{//1x1 squeeze}\end{aligned} \tag{1}$$

For IdleBlock, the theoretical computation complexity is

$$\begin{aligned}\sum(2 \cdot r \cdot C^2 \cdot (1-\alpha) \cdot HW, \quad & \text{//1x1 expand} \\ 2 \cdot r \cdot C \cdot (k^2 \cdot \frac{HW}{s^2}), \quad & \text{//k by k depthwise} \\ 2 \cdot r \cdot C^2 \cdot (1-\alpha) \cdot \frac{HW}{s^2}) \quad & \text{//1x1 squeeze}\end{aligned} \tag{2}$$

By replacing a MBBlock with an equivalent IdleBlock, we save:

$$2 \cdot \alpha \cdot r \cdot C^2 \cdot (HW + \frac{HW}{s^2}) \tag{3}$$

3. Hybrid Composition Networks

In the previous section we introduced the IdleBlock. In this section we introduce Hybrid Composition (HC), a novel non-monotonic network composition method.

In hybrid composition, at each stage of the network, we use a non-monotonic composition of multiple types building blocks. This is only possible if the different blocks have identical constraints on the input and output dimensions. For example, ShuffleBlock v1/v2 can be hybridized with a Bottleneck block, because both of the blocks accept a narrow input and output an expanded tensor. A Bottleneck block is not able to be hybridized with MBBlock, because these two blocks contains different bottleneck designs and the input and output dimensions are very different. Hybrid composition attempts to jointly utilize the different properties of multiple building blocks, which monotonically design is not able to do.

For a monotonic MBBlock network, such as MobileNet v2/v3 and EfficientNet, the first pointwise convolution operator in MBBlock is only used to expand the input dimension for the depthwise operation.

In our case with IdleBlock, both IdleBlock and MBBlock satisfy the input and output constraints for hybrid composition. Moreover, once we hybridize IdleBlock with MBBlock, the first pointwise convolution operator in MBBlock will be able to help us exchange information of two branches in IdleBlock, without an explicit channel shuffle operation as in ShuffleBlock.

However, hybrid composition introduces another challenge. If a network stage contains n MBBlock units, there are 2^n candidate combinations of MBBlock and IdleBlock placements in the Idle network, but we seek to explore only explore a small subset of these candidates.

To solve the challenge, we explore three configurations for the hybrid composition of MBBlock with IdleBlock: Maximum, None and Adjacent.

Maximum configuration

We only use MBBlock as reduction block, or when the network changes the output dimension. All other MBBlocks will be replaced by IdleBlock. This is the computational lower bound of hybrid composition with IdleBlock, as we need at least one MBBlock unit to exchange information in each stage.

None configuration

In this configuration, none of any blocks in the stage will be replaced with IdleBlock. This is the the computational upper bound of hybrid composition with IdleBlock.

Adjacent configuration

Adjacent configuration is a greedy strategy. We iteratively replace each MBBlock that has an MBBlock input with an IdleBlock, stopping when we reach our specified computational budget.

When we set $\alpha = 0.5$ in the IdleBlock during the repeated stage (stride=1), we notice that according to Equation 3, the theoretical computation cost of one MBBlock is roughly equal to cost of two IdleBlocks. This means that in the Adjacent configuration, to keep same computation complexity, we can replace one MBBlock with two IdleBlocks — resulting in an approximately 30% increase in overall network depth with same computation budget. We denote this special case as ”Adjacent +1 IdleBlock” configuration.

Intuitively, a neural architecture search over the space of block placement policies should fall (in terms of accuracy and computation cost) between the lower bound and upper bounds described here (Maximum and None).

4. Experiments

We present experimental results on ImageNet 2012 classification dataset [6] to demonstrate the effectiveness of hybrid composition with IdleBlocks. We set pruning factor α to 0.5 for all our experiments. We focus on studying MobileNet v3 and EfficientNet-B0, two state-of-art efficient image recognition architectures. We report execution times on Intel Xeon CPUs and ARM Cortex CPUs. We also conduct ablation studies to validate our design decisions. In this section, we use ”Top-1” to refer ”Top-1 Accuracy” on ImageNet.

4.1. Training Setup

We use MXNet [2] and GluonCV [7] for both single node training and distributed training. Our source code and pretrained models can be found at: URL.

Single Node

We provide MobileNet v3 experiment results on single node with 8 NVIDIA V100 GPU. We use fp16/32 mixed-precision during training. The batch size is set to 256 for each GPU (2048 in total). We use a Nesterov optimizer with learning rate set to 2.6 and weight decay set to 3.0e-5 for all models. All models are trained with 360 epoches. We warm up training in the first 5 epochs.

Distributed

We provide both MobileNet v3 and EfficientNet-B0 experiment results in distributed setting. All models are trained with 8 nodes, contains 64 V100 in total. Without special note, batch size is set to 128 per GPU. We use LB-SGD [15] to optimize all models, with initial learning rate 0.8, momentum 0.9, L2 weight decay 3.0e-5. All models are trained with 80 epochs (equivalent to 640 epoches on single node). We warm up training in the first 5 epoches. All distributed models are trained with Mixup [29] augmentation.

4.2. Inference Setup

We evaluate the inference time on Google Cloud Engine in single threaded and multi-threaded (4 threads) settings, on an Intel SkyLake CPU @ 2.0GHz. For ARM Cortex CPUs, we evaluate on an ARM Cortex-A53 @ 600MHz. All models are compiled with TVM [3] and AutoTVM [4], which provides substantial latency improvements on these architectures.

4.3. Result on MobileNet v3

We first study hybrid composition with IdleBlock on MobileNet v3. We evaluate different hybridizing configurations (Table 1). With HC and $\alpha = 0.5$ IdleBlock, if we seek to prune the network, we can prune up to 23% of the MAdds with only approximately 0.7 percentage point Top-1 accuracy loss. If we seek similar MAdds, we can improve network accuracy by approximately 0.4 to 0.9 percentage point under different training settings. If we go deeper with HC, the deeper networks are potentially more efficient than all state-of-art human expert-designed network, and all state-of-art searched networks (Table 2).

4.4. Result on EfficientNet-B0

Similar to the MobileNet-v3 experiment, we experiment with different configurations of hybrid composition on EfficientNet-B0 (Table 3). With HC and $\alpha = 0.5$ IdleBlock, if we seek to prune the network, we can prune up to 22% of the MAdds with only approx 0.7 percentage point Top-1 accuracy loss, which is similar to observations on MobileNet v3. If we seek similar MAdds, we can improve network accuracy by approx 0.6 percentage point. If we go deeper with HC, the deeper networks are consistently more efficient. (Table 2).

4.5. Inference Result on x86/ARM CPU

Our current implementation, based on TVM, realizes approximately one-third of the theoretical speedups obtained from our new block structure across a range of hardware. Future work may involve representing the entire block structure in the tensor expression IR, which will allow more aggressive optimizations such as completely eliminating the concatenation, splitting, and various layout transformation operations that are present in our implementation. Nevertheless, we still obtain real speed up with same depth without these specialized optimization on very different hardware platforms (Figure 10, Figure 11).

4.6. Ablation Study

4.6.1 Impact of Channel Shuffle Operator

In Section 2 we mentioned that the channel shuffle operator is not friendly to heterogeneous hardware accelerators for inference, despite its minimal theoretical computation complexity. In this experiment, we want to quantitatively understand channel shuffle operators contribution in network accuracy. We refer IdleBlock with channel shuffle operator as Inverted Shuffle Block (ISB). The difference between ISB and ShuffleBlock v2 is: ISB is based on MBBlock while ShuffleBlock v2 is based on a special case of Bottleneck block.

Config	MAdds (M)	Params (M)	#MBBlock	#IdleBlock	Top-1 (%)	ARM 1T/4T (ms)	x86 1T/4T (ms)
None (baseline)	219.4	6.44	15	0	75.20/75.11*	780/230	20.1/10.1
Maximum	**168.5**	5.51	5	10	74.67/74.42*	685/201	19.5/10.3
Adjacent	185.9	5.64	10	5	75.03/74.86*	711/207	18.9/9.9
Adjacent + 1 IdleBlock	224.4	6.06	10	10	75.58/75.82*	844/246	22.0/11.9
Adjacent + 1 IdleBlock L/R	224.4	6.06	10	10	**75.66/76.04***	844/245	22.2/11.8

Table 1: Applying different Hybrid Composition configurations on MobileNet-v3. $\star$ indicates using distributed training. The None configuration is the standard MobileNet v3. Adjacent + 1 IdleBlock L/R is the configuration where we replace one MBBlock with one L-IdleBlock and one R-IdleBlock. When adding or substituting MBBlock with IdleBlock, we use the same SE [11], channels, and activation settings from the substituted MBBlock.

Model	MAdds (M)	Params (M)	Top-1
MobileNet v3 [9]	219	5.4	75.2
FBNet-A [26]	249	4.3	73.0
FBNet-B [26]	295	4.5	74.1
MobileNet v2 [21]	300	3.4	72.0
Proxyless [1]	320	4.1	74.6
FBNet-C [26]	375	5.5	74.9
EfficientNet-B0 [24]	390	5.3	76.3
NASNet-B [31]	488	5.3	72.5
NASNet-C [31]	558	4.9	74.5
NASNet-A [31]	564	5.3	74.0
RandomWired-WS [28]	583	5.6	74.7
AmoebaNet [20]	580	6.4	75.7
MobileNet v2 (1.4) [21]	585	6.9	74.7
PNAS [17]	588	5.1	74.2
ShuffleNet v2 [19]	591	7.4	74.9
DARTS [18]	595	4.9	73.1
HC(M=5, I=10)	169	5.5	74.7
HC(M=10, I=10)	224	6.0	75.7
HC(M=15, I=10)	299	7.3	76.8
HC(M=15, I=20)	380	8.1	77.2
HC(M=5, I=10)*	169	5.5	74.4
HC(M=10, I=10)*	224	6.0	76.0
HC(M=15, I=10)*	299	7.3	77.0
HC(M=15, I=20)*	380	8.1	77.5

Table 2: A comparison of Hybrid Composition with IdleBlock on MobileNet v3 with state-of-the-art human expert-designed & NAS networks. Our work is listed as HC(M=x, I=y): M is total count of MBBlock and I is total count of IdleBlock. Models trained in a distributed setting are denoted by ($\star$).

We use MobileNet-v3 to study this problem. First we replace 14 out of 15 MBBlock to ISB or IdleBlock, only keep the very first MBBlock. We also tried different hybrid composition configurations in Table 2. Detailed results can be found in Table. 5. This experiment suggests that channel shuffle operators are unnecessary in our design.

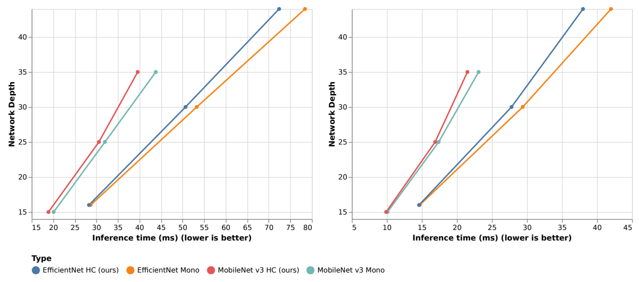

Figure 10: x86 Inference on network with different depth.Left: 1 Thread; Right: 4 Threads

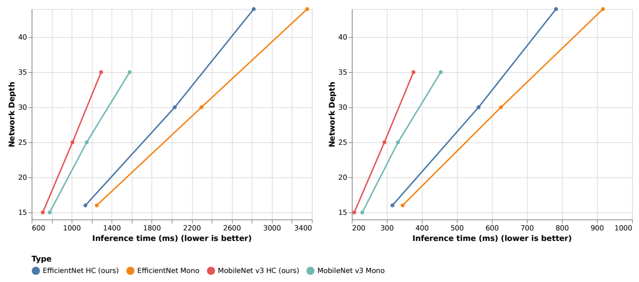

Figure 11: ARM Inference on network with different depth. Left: 1 Thread; Right: 4 Threads

4.6.2 Deeper Net with Monotonous MBBlock vs Hybrid Composition with IdleBlock

By applying hybrid composition with IdleBlock, we can save computational cost for a fixed depth. In this experiment, we want to understand the the accuracy difference between monotonous stacking with MBBlock vs hybrid composition with IdleBlock when going deeper. We study on both MobileNet v3 and EfficientNet-B0.

From Table 6, we find stacking more MBBlock monotonically does not always guarantee better accuracy with different training methods. Stacking MBBlock usually costs 1.3X more MAdds to achieve the same depth, compared to hybrid composition with IdleBlock. The accuracy

Config	MAdds (M)	Params (M)	#MBBlock	#IdleBlock	Top-1 (%)	ARM 1T/4T (ms)	x86 1T/4T (ms)
None	385.3	5.07	16	0	77.05	1250/345	28.7/14.7
Maximum	299.7	4.21	7	9	76.34	1066/298	27.2/13.8
Adjacent	334.4	4.54	11	5	76.61	1137/316	28.3/14.6
Adjacent + 1 IdleBlock	397.81	5.46	11	10	**77.60**	1373/384	32.2/17.2
Adjacent + 1 IdleBlock L/R	397.81	5.46	11	10	77.25	1372/383	32.1/17.4

Table 3: Applying different Hybrid Composition configurations on EfficientNet-B0. As in the MobileNet v3 experiments, SE, channels, non-linear activations and DropConnect [25] settings are set to be identical to the substituted MBBlock.

Model	MAdds (M)	Params (M)	Top-1
EfficientNet-B0 [24]	390	5.3	76.3
EfficientNet-B1 [24]	700	7.8	78.8
EfficientNet-B2 [24]	1000	9.2	79.8
EfficientNet-B3 [24]	1800	12	81.1
ResNet-50 v1$^\diamond$	4100	26.0	77.3
ResNeXt-50 32x4$^\diamond$	4200	25.0	79.3
ResNeXt-101 32x4$^\diamond$	7800	44.3	80.3
HRNet-W32-C [23]	8310	41.2	79.5
ResNet-152 v1$^\diamond$	11000	60.0	79.2
Squeeze-Excite-Net [11]	21000	115.1	81.3
EfficientNet-B0*	385	5.1	77.0
HC(M=16, I=14)	569	7.1	79.0
HC(M=16, I=28)	752	9.2	79.5
HC(M=16, I=28)$^\square$	1532	9.2	81.0

Table 4: A comparison of Efficient-B0 with hybrid composition to state-of-the-art models. ($\star$) indicates our implementation. ($^\diamond$) indicates results from GluonCV [7], which improves the results in the original paper due to the use of Mixup and other advanced training methods. ($^\square$) indicates the network is trained & tested with images at 320×320 resolution.

Model	Madds	Params	Top-1
M=1, ISB=14	157.8	4.8	73.8/73.7*
M=1, Idle=14	157.8	4.8	**73.8/73.8***
HC(M=10, ISB=10)	299.8	7.3	76.4/76.6*
HC(M=10, Idle=10)	299.8	7.3	**76.8/76.7***
HC(M=15. ISB=20)	380.1	8.1	77.0/77.2*
HC(M=15, Idle=20)	380.1	8.1	**77.2/77.4***

Table 5: Ablation study on impact of channel shuffle operator. We adopt block property settings from MobileNet v3. ($\star$) denotes number from distributed training.

difference is approximately 0.3 percentage points at various depth settings.

MboileNet v3	MAdds (M)	Params (M)	Top-1 (%)
M=15, I=10	299.8	7.26	76.8/**77.0***
M=25, I=0	368.4	8.85	76.9/76.7*
M=15, I=20	380.1	8.09	77.2/77.5*
M=35, I=0	517.5	11.26	**77.6**/77.2*
EfficientNet B0	MAdds (M)	Params (M)	Top-1 (%)
M=16, I=14	569.0	7.12	79.0
M=30, I=0	715.94	8.32	**79.2**
M=16, I=28	752.2	9.18	79.5
M=44, I=0	1046.0	11.57	**79.8**

Table 6: Ablation study on deep monotonous networks with MBBlock and deep hybrid composition networks with IdleBlock. In this table M indicates count of MBBlock, and I indicates count of IdleBlock. ($\star$) denotes number from distributed training.

4.6.3 Impact of L/R-IdleBlock

Empirically, for shallow networks, a larger receptive field will improve accuracy. We use MobileNet v3 to study this phenomena. Result is listed in Table 7. We find L/R-IdleBlock placement has an impact on the final accuracy. Most of our experiments empirically validate our hypothesis, as monotonically stacked IdleBlock create a larger receptive field. In one of one settings, L/R cross placement performs better than monotonically use IdleBlock. L/R-Idle block placement introduce a new hyperparameter in network design, but this hyperparameter is substantially less sensitive than other hyper-parameters.

5. Conclusion and Future work

In this paper, we introduce a new network design methodology inspired by network pruning: Idle and its MBBlock version IdleBlock. In order to utilize IdleBlock efficiently, we break the tradition of monotonous design in network stage and introduce hybrid composition of IdleBlock with MBBlock. **Our empirical results suggest that using the pruned computation to make the network deeper with hybrid composition is more efficient than**

MobileNet v3	Top-1	Dist. Top-1
M=1, L/R-Idle=14	73.09	73.48
M=1, Idle=14	**73.82**	**73.78**
M=5, L/R-Idle=10	74.23	74.42
M=5, Idle=10	**74.67**	74.42
M=15, L/R-Idle=20	76.38	**77.57**
M=15, Idle=20	**77.20**	77.39

Table 7: Ablation study on monotonically use IdleBlock or mix use L/R-IdleBlock with different depth settings under MobileNet v3. In this table M indicates number of MB-Block, L/R-IdleBlock indicates using L-IdleBlock and R-IdleBlock alternately.

various state-of-art neural architecture search methods for small models. Hybrid composition also makes intermediate and large networks more efficient. This may point to a simpler direction for future neural architecture search directions for more efficient image recognition network. The principles of Idle design may also be worth to extending to neural networks for other tasks and domains.

References

[1] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 7

[2] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015. 6

[3] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated end-to-end optimizing compiler for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 578–594, 2018. 6

[4] Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Learning to optimize tensor programs. In *Advances in Neural Information Processing Systems*, pages 3389–3400, 2018. 6

[5] Yunpeng Chen, Haoqi Fang, Bing Xu, Zhicheng Yan, Yannis Kalantidis, Marcus Rohrbach, Shuicheng Yan, and Jiashi Feng. Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution. *arXiv preprint arXiv:1904.05049*, 2019. 1

[6] 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. 3, 6

[7] Jian Guo, He He, Tong He, Leonard Lausen, Mu Li, Haibin Lin, Xingjian Shi, Chenguang Wang, Junyuan Xie, Sheng Zha, et al. Gluoncv and gluonnlp: Deep learning in computer vision and natural language processing. *arXiv preprint arXiv:1907.04433*, 2019. 6, 8

[8] 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. 1, 3

[9] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. *arXiv preprint arXiv:1905.02244*, 2019. 1, 7

[10] 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. 1, 3

[11] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 7, 8

[12] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 1

[13] Tsung-Wei Ke, Michael Maire, and Stella X Yu. Multigrid neural architectures. In *Proceedings of the IEEE Conference*

on Computer Vision and Pattern Recognition, pages 6665–6673, 2017. 1

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1

[15] Haibin Lin, Hang Zhang, Yifei Ma, Tong He, Zhi Zhang, Sheng Zha, and Mu Li. Dynamic mini-batch sgd for elastic distributed training: Learning in the limbo of resources. *arXiv preprint arXiv:1904.12043*, 2019. 6

[16] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. 1

[17] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018. 7

[18] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 7

[19] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018. 1, 7

[20] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019. 7

[21] 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. 1, 3, 7

[22] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[23] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. *arXiv preprint arXiv:1902.09212*, 2019. 1, 8

[24] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 1, 7, 8

[25] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066, 2013. 8

[26] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019. 7

[27] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 1

[28] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. *arXiv preprint arXiv:1904.01569*, 2019. 7

[29] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 6

[30] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018. 1

[31] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 7