Rethinking Bottleneck Structure for Efficient Mobile Network Design

Daquan Zhou[1,2,3*†], Qibin Hou[1*], Yunpeng Chen[2], Jiashi Feng[1], and Shuicheng Yan[2]

[1] National University of Singapore, Singapore
[2] Yitu Technology
[3] Institute of Data Science, NUS
{zhoudaquan21,andrewhoux}@gmail.com, elefjia@nus.edu.sg,
{yunpeng.chen, shuicheng.yan}@yitu-inc.com

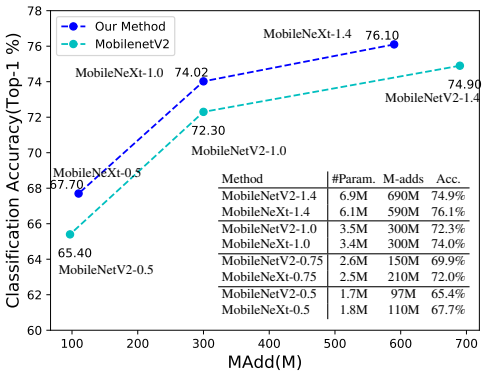
Abstract. The inverted residual block is dominating architecture design for mobile networks recently. It changes the classic residual bottleneck by introducing two design rules: learning inverted residuals and using linear bottlenecks. In this paper, we rethink the necessity of such design changes and find it may bring risks of information loss and gradient confusion. We thus propose to flip the structure and present a novel bottleneck design, called the sandglass block, that performs identity mapping and spatial transformation at higher dimensions and thus alleviates information loss and gradient confusion effectively. Extensive experiments demonstrate that, different from the common belief, such bottleneck structure is more beneficial than the inverted ones for mobile networks. In ImageNet classification, by simply replacing the inverted residual block with our sandglass block without increasing parameters and computation, the classification accuracy can be improved by more than 1.7% over MobileNetV2. On Pascal VOC 2007 test set, we observe that there is also 0.9% mAP improvement in object detection. We further verify the effectiveness of the sandglass block by adding it into the search space of neural architecture search method DARTS. With 25% parameter reduction, the classification accuracy is improved by 0.13% over previous DARTS models. Code can be found at: https://github.com/zhoudaquan/rethinking_bottleneck_design.

Keywords: sandglass block; residual block; efficient architecture design; image classification

1 Introduction

A common belief behind the design principles of most popular light-weight models (either manually designed or automatically searched) [27, 31, 34, 35] is to adopt the inverted residual block [31]. Compared to the classic residual bottleneck block [12, 14], this block shifts the identity mapping from high-dimensional representations to low-dimensional ones (*i.e.,* the bottlenecks). However, connecting identity mapping between thin bottlenecks would inevitably lead to information loss since the residual representations are compressed as shown in Figure 2(b). Moreover, it would also weaken the

* Authors contributed equally.
† Work done during an internship at Yitu Tech.



Method	#Param.	M-adds	Acc.
MobileNetV2-1.4	6.9M	690M	74.9%
MobileNeXt-1.4	6.1M	590M	76.1%
MobileNetV2-1.0	3.5M	300M	72.3%
MobileNeXt-1.0	3.4M	300M	74.0%
MobileNetV2-0.75	2.6M	150M	69.9%
MobileNeXt-0.75	2.5M	210M	72.0%
MobileNetV2-0.5	1.7M	97M	65.4%
MobileNeXt-0.5	1.8M	110M	67.7%

Fig. 1. Top-1 classification accuracy comparisons between the proposed MobileNeXt and MobileNetV2 [31]. We use different width multipliers to trade-off between model complexity and accuracy. Here, four widely-used multipliers are chosen, including 0.5, 0.75, 1.0, and 1.4. As can be seen, under each width multiplier, our MobileNeXt surpasses the MobileNetV2 baseline by a large margin, especially for the models with less learnable parameters.

propagation capability of gradients across layers, due to gradient confusion arising from the narrowed feature dimensions [32]. Therefore, despite the wide use of the inverted residual block, how to design residual blocks for mobile devices is worthy of studying.

In this paper, in view of the above concerns, we rethink the rationality of shifting from the classic bottleneck structure (Figure 2(a)) to the popular inverted residual block (Figure 2(b)) in developing mobile networks. In particular, we consider the following three fundamental questions. (i) What are the effects if we position the identity mapping (*i.e.,* shortcuts) at the high-dimensional representations as done in the classic bottleneck structure? (ii) While the linear activation can reduce information loss, should it only be applied to the bottlenecks? (iii) The previous questions remind us of the classic bottleneck structure which suffers high computational complexity. This cost can be reduced by replacing the dense spatial convolutions with depthwise ones, but, regarding the bottlenecks, should the depthwise convolution be still added in the low-dimensional bottleneck as conventional?

Motivated by the above questions, we present and evaluate a new bottleneck design, termed the sandglass block. Unlike the inverted residual block that builds shortcuts between linear bottlenecks, our sandglass block puts shortcut connections between linear high-dimensional representations, as shown in Figure 2(c). Such structure preserves more information delivered between blocks compared to the inverted residual block and propagates more gradients backward to better optimize network training because of the high-dimensional residuals [32]. Furthermore, to learn more expressive spatial representation, instead of putting the spatial convolutions in the bottleneck with compressed channels, we propose to apply them in the expanded high dimensional feature space, which we find is an effective way of improving the model performance. In addition, we maintain the channel reduction and expansion process with pointwise convolutions to reduce computational cost. This makes our block quite different from the inverted residual block but more similar to the classic residual bottleneck.

We stack the sandglass blocks in a modularized way to build the proposed MobileNeXt. Our network achieves more than 1.7% top-1 classification accuracy improvement over MobileNetV2 on ImageNet with slightly less computation and a comparable

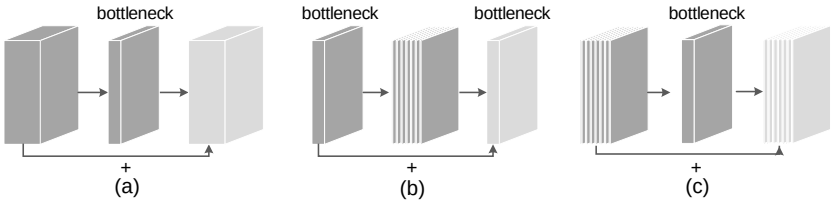


Fig. 2. Conceptual diagram of different residual bottleneck blocks. (a) Classic residual block with bottleneck structure [13]. (b) Inverted residual block [31]. (c) Our proposed sandglass block. We use thickness of each block to represent the corresponding relative number of channels. As can be seen, compared to the inverted residual block, the proposed residual block reverses the thought of building shortcuts between bottlenecks and adds depthwise convolutions (detached blocks) at both ends of the residual path, both of which are found crucial for performance improvement.

number of parameters as shown in Figure 1. When applying the sandglass block on the EfficientNet topology to replace the inverted residual block, the resulting model surpasses the previous state-of-the-art by 0.5% with a comparable amount of computation but 20% parameter reduction. Particularly, in object detection, when taking SSDLite [25, 31] as the object detector, using our MobileNeXt as backbone gains 0.9% in mAP on the Pascal VOC 2007 test set over MobileNetV2. More interestingly, we also experimentally find the proposed sandglass block can be used to enrich the search space of neural architecture search algorithms [24]. By adding the sandglass block into the search space as a 'super' operator, without changing the search algorithm, the resultant model can improve classification accuracy by 0.13% but with 25% less parameters compared to models searched from the vanilla space.

In summary, we make the following contributions in this paper:

- Our results advocate a rethinking of the bottleneck structure for mobile network design. It seems that the inverted residuals are not so advantageous over the bottleneck structure as commonly believed.
- Our study reveals that building shortcut connections along higher-dimensional feature space could promote model performance. Moreover, depthwise convolutions should be conducted in the high dimensional space for learning more expressive features and learning linear residuals is also crucial for bottleneck structure.
- Based on our study, we propose a novel sandglass block, which substantially extends the classic bottleneck structure. We experimentally demonstrate that this structure is more suitable for mobile applications in terms of both accuracy and efficiency and can be used as 'super' operators in architecture search algorithms for better architecture generation.

2 Related Work

Modern deep neural networks are mostly built by stacking building blocks, which are designed based on either the classic residual block with bottleneck structure [12] or the

inverted residual block [31]. In this section, we categorize all related networks based on above two types of building blocks and briefly describe them below.

Classic residual bottleneck blocks The bottleneck structure was first introduced in ResNet [12]. A typical bottleneck structure consists of three convolutional layers: an 1×1 convolution for channel reduction, a 3×3 convolution for spatial feature extraction, and another 1×1 convolution for channel expansion. A residual network is often constructed by stacking a sequence of such residual blocks. The bottleneck structure was further developed in later works by widening the channels in each convolutional layer [41], applying group convolutions to the middle bottleneck convolution for aggregating richer feature representations [39], or introducing attention based modules to explicitly model inter-dependencies between channels [18, 23]. There are also other works [3, 37] combining residual blocks with dense connections to boost the performance. However, in spite of the success in heavy-weight network design, it is rarely used in light-weight networks due to the model complexity. Our work demonstrates that by reasonably adjusting the residual block, this kind of classic bottleneck structure is also suitable for light-weight networks and can yield state-of-the-art results.

Inverted residual blocks The inverted residual block, which was first introduced in MobileNetV2 [31], reverses the idea of the classic bottleneck structure and connects shortcuts between linear bottlenecks. It largely improves performance and optimizes the model complexity compared to the classic MobileNet [17] which is composed of a sequence of 3×3 depthwise separable convolutions. Because of high efficiency, the inverted residual block has been widely adopted in the later mobile network architectures. ShuffleNetV2 [27] inserts a channel split module before the inverted residual block and adds another channel shuffle module after it. In HBONet [22], down-sampling operations are introduced into inverted residual blocks for modeling richer spatial information. MobileNetV3 [16] proposes to search for optimal activation functions and the expansion rate of inverted residual blocks at each stage. More recently, MixNet [36] proposes to search for optimal kernel sizes of the depthwise separable convolutions in the inverted residual block. EfficientNet [35] is also based on the inverted residual block but differently it uses a scaling method to control the network weight in terms of input resolution, network depth, and network width. Different from all the above approaches, our work advances the standard bottleneck structure and demonstrates the superiority of our building block over the inverted residual block in mobile settings.

Model compression and neural architecture search Model compression algorithms are effective for removing redundant parameters for neural networks, such as network pruning [2, 11, 26, 30], quantization [5, 19], factorization [20, 43], and knowledge distillation [15]. Despite efficient networks, the performance of the compressed networks is still closely related to the original networks' architectures. Thus, designing more efficient network architectures is essential for yielding efficient models. Neural architecture search achieves so by automatically searching efficient network architectures [1, 9, 34] or even parameters [42]. However, the search space (SS) requires human expertise and the performance of the searched networks is largely dependent upon SS as pointed out in [6, 40]. In this paper, we show that our proposed building block is complementary

to existing SS design principles and can further improve the performance of searched networks if added to existing search spaces.

3 Method

In this section, we first review preliminaries about the bottleneck structure used in previous residual networks and then describe our proposed block and network architecture.

3.1 Preliminaries

Residual block with bottleneck structure The classic residual block with bottleneck structure [12], as shown in Figure 2(a), consists of two 1×1 convolution layers for channel reduction and expansion respectively and one 3×3 convolution layer between them for spatial information encoding. In spite of its success in heavy-weight network design [12], this conventional bottleneck structure is not suitable for building light-weight neural networks because of its large amount of parameters and computation cost in the standard 3×3 convolutional layer.

Depthwise separable convolutions To reduce computational cost and make the network more efficient, depthwise separable convolutions [4, 17] are developed to replace the standard one. As demonstrated in [4], a convolution with a $k \times k \times M \times N$ weight tensor, where $k \times k$ is the kernel size and M and N are the number of input and output channels respectively, can be factorized into two convolutions. The first is an M-channel $k \times k$ depthwise (*a.k.a* channel-wise) convolution to learn the spatial correlations among locations within each channel separately. The second is a pointwise convolution that learns to linearly combine channels to produce new features. As the combination of a pointwise convolution and a $k \times k$ depthwise convolution has significantly less parameters and computations, using depthwise separable convolutions in basic building blocks can remarkably reduce the parameters and computational cost. Our proposed architecture also adopts such separable convolutions.

Inverted residual block The inverted residual block is specifically tailored for mobile devices, especially those with limited computational resource budget. More specifically, unlike the classic bottleneck structure as shown in Figure 2(a), to save computations, it takes as input a low-dimensional compressed tensor and expands it to a higher dimensional one by a pointwise convolution. Then it applies depthwise convolution for spatial context encoding, followed by another pointwise convolution to generate a low-dimensional feature tensor as input to the next block. The inverted residual block presents two distinct architecture designs for gaining efficiency without suffering too much performance drop: the shortcut connection is put between the low-dimensional bottlenecks if necessary (as shown in Figure 2(b)); and linear bottleneck is adopted.

Despite good performance [31], in inverted residual blocks, feature maps encoded by the intermediate expansion layer should be first projected to low-dimensional ones, which may not preserve enough useful information due to channel compression. Moreover, recent studies have unveiled that wider architecture is more favorable for alleviating gradient confusion [32] and hence can improve network performance. Putting

shortcut connections between bottlenecks may prevent the gradients from top layers from being successfully propagated to bottom layers during model training because of the low-dimensionality of representations between adjacent inverted residual blocks.

3.2 Sandglass Block

In view of the aforementioned limitations of the inverted residual block, we rethink its design rules and present a sandglass block that can tackle the above issues by flipping the thought of inverted residuals.

Our design principle is mainly based on the following insights: (i) To preserve more information from the bottom layers when transiting to the top layers and to facilitate the gradients propagation across layers, the shortcuts should be positioned to connect high-dimensional representations. (ii) Depthwise convolutions with small kernel size (*e.g.,* 3×3) are light-weight, so we can appropriately apply a couple of depthwise convolutions onto the higher-dimensional features such that richer spatial information can be encoded to generate more expressive representations. We elaborate on these design considerations in the following.

Rethinking the positions of expansion and reduction layers Originally, the inverted residual block performs expansion at first and then reduction. Based on the aforementioned design principle, to make sure the shortcuts connect high-dimensional representations, we propose to reverse the order of the two pointwise convolutions first. Let $\mathbf{F} \in \mathbb{R}^{D_f \times D_f \times M}$ be the input tensor and $\mathbf{G} \in \mathbb{R}^{D_f \times D_f \times M}$ the output tensor of a building block[4]. We do not consider the depthwise convolution and activation layers at this moment. The formulation of our building block can be written as follows:

$$\mathbf{G} = \phi_e(\phi_r(\mathbf{F})) + \mathbf{F}, \tag{1}$$

where ϕ_e and ϕ_r denote the two pointwise convolutions for channel expansion and reduction, respectively. In this way, we can keep the bottleneck in the middle of the residual path for saving parameters and computation cost. More importantly, this allows us to use the shortcut connection to connect representations with a large number of channels instead of the bottleneck ones.

High-dimensional shortcuts Instead of putting shortcuts between bottlenecks, we put shortcuts between higher-dimensional representations as shown in Figure 3(b). The 'wider' shortcut delivers more information from the input $\mathbf{F}$ to the output $\mathbf{G}$ compared to the inverted residual block and allows more gradients to propagate across multiple layers.

Learning expressive spatial features Pointwise convolutions can be used to encode the inter-channel information but fail to capture spatial information. In our building block, we follow previous mobile networks and adopt depthwise spatial convolutions. The inverted residual block adds depthwise convolutions between pointwise convolutions to

[4] For simplicity, we assume that the input and output of the building block share the same number of channels and resolution.

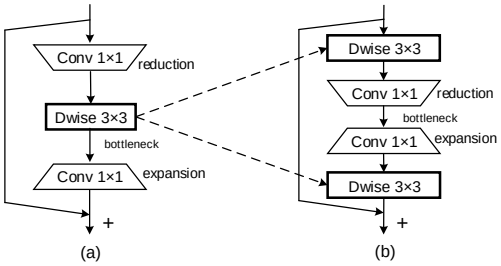


Fig. 3. Different types of residual blocks. (a) Classic bottleneck structure with depthwise spatial convolutions. (b) Our proposed sandglass block with bottleneck structure. To encode more expressive spatial information, instead of adding depthwise convolutions in the bottleneck, we propose to move them to the ends of the residual path with high-dimensional representations.

Table 1. Basic operator description of the proposed sandglass block. Here, 't' and 's' denote the channel reduction ratio and the stride, respectively.

Input dimension	Operator type	Output dimension
$D_f \times D_f \times M$	3×3 Dwise conv, ReLU6	$D_f \times D_f \times M$
$D_f \times D_f \times M$	1×1 conv, linear	$D_f \times D_f \times \frac{M}{t}$
$D_f \times D_f \times \frac{M}{t}$	1×1 conv, ReLU6	$D_f \times D_f \times N$
$D_f \times D_f \times N$	3×3 Dwise conv, linear, stride = s	$\frac{D_f}{s} \times \frac{D_f}{s} \times N$

learn expressive spatial context information. However, in our case, the position between two pointwise convolutions is the bottleneck. Directly adding depthwise convolutions in the bottleneck as shown in Figure 3(a) makes them have fewer filters and thus, less spatial information can be encoded. We experimentally found that this structure largely degrades the performance compared to MobileNetV2 by more than 1%.

Regarding the positions of the pointwise convolutions, instead of directly putting the depthwise convolution between the two pointwise convolutions, we propose to add depthwise convolutions at the ends of the residual path as shown in Figure 3(b). Mathematically, our building block can be formulated as follows:

$$\hat{\mathbf{G}} = \phi_{1,p}\phi_{1,d}(\mathbf{F}) \tag{2}$$

$$\mathbf{G} = \phi_{2,d}\phi_{2,p}(\hat{\mathbf{G}}) + \mathbf{F} \tag{3}$$

where $\phi_{i,p}$ and $\phi_{i,d}$ are the i-th pointwise and depthwise convolutions, respectively. In this way, since both depthwise convolutions are conducted in high-dimensional spaces, richer feature representations can be extracted compared to the inverted residual block. We will give more explanations on the advantages of such design.

Activation layers It has been demonstrated in [31] that using linear bottlenecks helps in preventing the feature values from being zeroed. Following this suggestion, we do not add any activation layer after the reduction layer (the first pointwise convolutional

layer). It should also be noted that though the output of our building block is high-dimensional, we empirically found adding an activation layer after the last convolution can negatively influence the classification performance. Therefore, activation layers are only added after the first depthwise convolutional layer and the last pointwise convolutional layer. We will give more explanations in our experiments on this.

Block structure Taking the above considerations, we design a novel residual bottleneck block. The structure details are given in Table 1, and the diagram can also be found in Figure 3(b). Note that when the input and output have different channel numbers, we do not add the shortcut connection and for blocks with stride 2 we remove the first depthwise convolution. For depthwise convolutions, we always use kernel size 3×3 as done in other works [12, 31]. We also utilize batch normalization and ReLU6 activation if necessary during training.

Relation to the inverted and classic residual blocks Albeit both architectures exploit the bottlenecks, the design intuition and the internal structure are quite different. Our goal is to demonstrate that the idea of building shortcut connections between high-dimensional representations as in the classic bottleneck structure [12] is suitable for light-weight networks as well. To the best of our knowledge, this is the first work that attempts to investigate the advantages of the classic bottleneck structure over the inverted residual block for efficient network design. On the other hand, we also attempt to demonstrate that adding depthwise convolutions to the ends of the residual path in our structure can encourage the network to learn more expressive spatial information and hence yield better performance. In our experiment section, we will show more numerical results and provide detailed analysis.

3.3 MobileNeXt Architecture

Based on our sandglass block, we develop a modularized architecture, MobileNeXt. At the beginning of our network, there is a convolutional layer with 32 output channels. After that, our sandglass blocks are stacked together. Detailed information about the network architecture can be found in Table 2. Following [31], the expansion ratio used in our network is set to 6 by default. The output of the last building block is followed by a global average pooling layer to transform 2D feature maps to 1D feature vectors. A fully-connected layer is finally added to predict the final score for each category.

Identity tensor multiplier The shortcut connections in residual blocks have been shown essential for improving the capability of propagating gradients across layers [12, 31]. According to our experiments, we find that there is no need to keep the whole identity tensor to combine with the residual path. To make our network more friendly to mobile devices, we introduce a new hyper-parameter—identity tensor multiplier, denoted by $\alpha \in [0, 1]$, which controls what portion of the channels in the identity tensor is preserved. For convenience, let ϕ be the transformation function of the residual path in our block. Originally, the formulation of our block can be written as $G = \phi(F) + F$. After applying the multiplier, our building block can be rewritten as

$$G_{1:\alpha M} = \phi(F)_{1:\alpha M} + F_{1:\alpha M}, \quad G_{\alpha M:M} = \phi(F)_{\alpha M:M}, \tag{4}$$

Table 2. Architecture details of the proposed MobileNeXt. Each row denotes a sequence of building blocks, which is repeated 'b' times. The reduction ratio used in each building block is denoted by 't'. The stride of the first building block in each stage is set to 2 and all the others are with stride 1. Each convolutional layer is followed by a batch normalization layer and the kernel size for all spatial convolutions is set to 3×3. We do not add identity mappings for those blocks have different input and output channels. We suppose there are totally k categories.

No.	t	Output dimension	s	b	Input dimension	Operator
1	-	$112 \times 112 \times 32$	2	1	$224 \times 224 \times 3$	conv2d 3x3
2	2	$56 \times 56 \times 96$	2	1	$112 \times 112 \times 32$	sandglass block
3	6	$56 \times 56 \times 144$	1	1	$56 \times 56 \times 96$	sandglass block
4	6	$28 \times 28 \times 192$	2	3	$56 \times 56 \times 144$	sandglass block
5	6	$14 \times 14 \times 288$	2	3	$28 \times 28 \times 192$	sandglass block
6	6	$14 \times 14 \times 384$	1	4	$14 \times 14 \times 288$	sandglass block
7	6	$7 \times 7 \times 576$	2	4	$14 \times 14 \times 384$	sandglass block
8	6	$7 \times 7 \times 960$	1	3	$7 \times 7 \times 576$	sandglass block
9	6	$7 \times 7 \times 1280$	1	1	$7 \times 7 \times 960$	sandglass block
10	-	$1 \times 1 \times 1280$	-	1	$7 \times 7 \times 1280$	avgpool 7x7
11	-	k	-	1	$1 \times 1 \times 1280$	conv2d 1x1

where the subscripts index the channel dimension.

The advantages of using α are mainly two-fold. First, after reducing the multiplier, the number of element-wise additions in each building block can be reduced. As pointed out in [27], the element-wise addition is time consuming. Users can choose a lower identity tensor multiplier to yield better latency with nearly no performance drop. Second, the number of memory access times can be reduced. One of the main factors that affect the model latency is the memory access cost (MAC). As the shortcut identity tensor is from the output of the last building block, its recurrent nature hints an opportunity to cache it on the chip in order to avoid the excessive off-chip memory access. Therefore, reducing the channel dimension of the identity tensor can effectively encourage the processors to store it in the cache or other faster memory near the processors and hence improve the latency. We will give more details on how this multiplier affects the performance and model latency in the experiment section.

4 Experiments

4.1 Experiment Setup

We adopt the PyTorch toolbox [29] to implement all our experiments. We use the standard SGD optimizer to train our models with both decay and momentum of 0.9 and the weight decay is 4×10^{-5}. We use the cosine learning schedule with an initial learning rate of 0.05. The batch size is set to 256 and four GPUs are used for training. Without special declaration, we train all the models for 200 epochs and report results on the ImageNet [21] for classification and Pascal VOC dataset [7] for object detection. We use distributed training with three epochs of warmup.

Table 3. Comparisons with MobileNetV2 using different width multipliers with input resolution 224×224. As can be seen, the smaller the multiplier is set to the better performance gain we achieve over MobileNetV2 with comparable latency (*e.g.,* 210ms for both models with width multiplier 1.0) tested on Google Pixel 4XL under the PyTorch environment setting.

No.	Models	Param. (M)	MAdd(M)	Top-1 Acc. (%)
1	MobileNetV2-1.40	6.9	690	74.9
2	MobileNetV2-1.00	3.5	300	72.3
3	MobileNetV2-0.75	2.6	150	69.9
4	MobileNetV2-0.50	2.0	97	65.4
5	MobileNetV2-0.35	1.7	59	60.3
6	MobileNeXt-1.40	6.1	590	76.1
7	MobileNeXt-1.00	3.4	300	74.0
8	MobileNeXt-0.75	2.5	210	72.0
9	MobileNeXt-0.50	2.1	110	67.7
10	MobileNeXt-0.35	1.8	80	64.7

4.2 Comparisons with MobileNetV2

In this subsection, we extensively study the advantages of our MobileNeXt over MobileNetV2 under various settings. Besides comparing performance of their full models (*i.e.,* , with weight multiplier of 1) for classification, we also compare their performance with other weight multipliers and quantization. This can help unveil the performance advantage of our model w.r.t. the full spectrum of model architecture configurations.

Comparison under different width multipliers We use the width multiplier as a scaling factor to trade off the model complexity and accuracy of the model as used in [16, 17, 31]. Here, we adopt five different multipliers, including 1.4, 1.0, 0.75, 0.5, and 0.35, to show the superiority of our network over MobileNetV2. As shown in Table 3 [5], our networks with different multipliers all outperform MobileNetV2 with comparable numbers of parameters and computations. The performance gain of our model over MobileNetV2 is especially high when the multiplier is small. This demonstrates that our model is more efficient since our model performance is much better at small sizes.

Comparison under post-training quantization Quantization algorithms are often used in real-world applications as a kind of effective compression tool with subtle performance loss. However, the performance of the quantized model is significantly affected by the original base model. We experimentally show that the MobileNeXt can achieve better performance than the MobileNetV2 when combined with the quantization algorithm. Here, we use a widely-used post-training linear quantization method introduced in [28]. We apply 8-bit quantization on both weights and activations as 8-bit is the most

[5] We also conduct latency measurements with TF-Lite on Pixel 4XL and the measured latency for MobileNeXt and MobileNetV2 are 66ms and 68 ms respectively.

Table 4. Performance of our proposed MobileNeXt and MobileNetV2 after post-training quantization. In bites configurations, 'W' denotes the number of bits used to represent the weights of the model and 'A' denotes the number of bits used to represent the activations.

Model	Precision (W/A)	Method	Top-1 Acc. (%)
MobileNetV2	INT8/INT8	Post Training Quant.	65.07
MobileNeXt	INT8/INT8	Post Training Quant.	$68.62_{+3.55}$
MobileNetV2	FP32/FP32	-	72.25
MobileNeXt	FP32/FP32	-	$74.02_{+1.77}$

Table 5. Performance of our proposed network and MobileNetV2 when adding the number of spatial convolutions (Dwise convs) in each building block. Obviously, our MobileNeXt performs much better than the improved MobileNetV2 with less learnable parameters and computational cost.

Method	#Dwise convs	Param. (M)	M-Adds (M)	Top-1 Acc. (%)
MobileNetV2	2 (middle)	3.6	340	73.02
MobileNeXt	2 (top, bottom)	3.5	300	74.02

common scheme used on hardware platforms. The results are shown in Table 4. Without quantization, our network improves MobileNetV2 by more than 1.7% in terms of top-1 accuracy. When the parameters and activations are quantized to 8 bits, our network outperforms MobileNetV2 by 3.55% under the same quantization settings. The reasons for this large improvement are two-fold. First, compared to MobileNetV2, we move the shortcut in each building block from low-dimensional representations to high-dimensional ones. After quantization, more informative feature representations can be preserved. Second, using more depthwise spatial convolutions can help preserve more spatial information, which we believe is beneficial to the classification performance.

Comparison with MobileNetV2 on structure As shown in Figure 3(b), our sandglass block contains two 3×3 depthwise convolutions for encoding rich spatial context information. To demonstrate the benefit of our model comes from our novel architecture rather than leveraging one more depthwise convolution or larger receptive field, in this experiment, we attempt to compare with an improved version of MobileNetV2 with one more depthwise convolution inserted in the middle of each inverted residual block. The results are shown in Table 5. Obviously, after adding one more depthwise convolution, the performance of MobileNetV2 increases to 73%, which is still far worse than ours (74%) with even more learnable parameters and complexity. This indicates that structurally our network does have an edge over MobileNetV2.

4.3 Comparison with State-of-the-Art Mobile Networks

To further verify the superiority of our proposed sandglass block over the inverted residual blocks, we add squeeze and excite modules into our MobileNeXt as done in [16,35].

We do not apply any searching algorithms on the architecture design and data augmentation policy. We directly take the EfficientNet-b0 architecture [35] and replace the inverted residual block with sandglass block with the basic augmentation policy. As shown in table 6, with a comparable amount of computation and 20% parameter reduction, replacing the inverted residual block with sandglass block results in 0.4% top-1 classification accuracy improvement on ImageNet-1k dataset.

4.4 Ablation Studies

In Sec. 4.2, we have shown the importance of connecting high-dimensional representations with shortcuts. Here, we study how other model design choices contribute to the model performance and efficiency, including the effect of using wider transformation, the importance of learning linear residuals, and the role of identity tensor multiplier.

Importance of using wider transformation As described in Sec. 3, we apply spatial transformation and shortcut connections to high-dimensional representations. To demonstrate the importance of such operations, we follow the inverted residual block to use the shortcuts to connect the bottleneck representations. This operation leads to an accuracy decrease of 1%, which indicates the advantage of using wider transformation.

Importance of linear residuals According to MobileNetV2 [31], its classification performance will be degraded when replacing the linear bottleneck with the non-linear one because of information loss. From our experiment, we obtain a more general conclusion. We find that though the shortcuts connect high-dimensional representations in our model, adding non-linear activations (ReLU6) to the last convolutional layer decreases the performance by nearly 1% compared to the setting using linear activations (no ReLU6). This indicates that learning linear residual (*i.e.,* adding no non-linear activation layer on the top of the residual path) is essential for light-weight networks with shortcuts connecting either expansion layers or reduction layers.

Effect of identity tensor multiplier Here, we investigate how the identity tensor multiplier (Sec. 3.3) would trades-off the model accuracy and latency. We use pytorch to generate the model and run it on Google Pixel 4XL. For each model, we measure the average inference time of 10 images as the final inference latency. As shown in Table 7, the reduction of the multiplier has subtle impacts on the classification accuracy. When half of the identity representations are removed, the performance has no drop but the latency is improved. When the multiplier is set to $1/6$, the performance decreases by 0.34%, but with further improvement in terms of latency. This indicates that introducing such a hyper-parameter does matter for balancing the model performance and latency.

4.5 Application for Object Detection

To explore the transferable capability of the proposed approach against MobileNetV2, in this subsection, we apply our classification model to the object detection task as pretrained models. We use both the proposed network and MobileNetv2 as feature extractors and report results on the Pascal VOC 2007 test set [8] following [25] using

Table 6. Comparisons with other state-of-the-art models. MobileNeXt denotes the model based on our proposed sandglass block and MobileNeXt† denotes the models with sandglass block and the SE module [18] added for a fair comparison with other state-of-the-art models such as EfficientNet. We do not apply any searching algorithms.

Models	Param. (M)	MAdd (M)	Top-1 Acc. (%)
MobilenetV1-1.0 [17]	4.2	575	70.6
MobilenetV2-1.0 [31]	3.5	300	72.3
MnasNet-A1 [34]	3.9	312	75.2
MobilenetV3-L-0.75 [16]	4.0	155	73.3
ProxylessNAS [1]	4.1	320	74.6
FBNet-B [38]	4.5	295	74.1
GhostNet-1.3 [10]	7.3	226	75.7
EfficientNet-b0 [35]	5.3	390	76.3
MobileNeXt-1.0	3.4	300	74.02
MobileNeXt-1.0†	3.94	330	76.05
MobileNeXt-1.1†	**4.28**	420	**76.7**

Table 7. Model performance and latency comparisons with different identity tensor multipliers. As can be seen, the latency can be improved by using lower identity tensor multipliers with only negligible sacrifice on the classification accuracy.

No.	Models	Tensor multiplier	Param. (M)	Top-1 Acc. (%)	Latency (ms)
1	MobileNeXt	1.0	3.4	74.02	211
2	MobileNeXt	$1/2$	3.4	74.09	196
3	MobileNeXt	$1/3$	3.4	73.91	195
4	MobileNeXt	$1/6$	3.4	73.68	188

SSDLite [25, 31]. Similar to [31], the first and second layers of SSDLite are connected to the last pointwise convolution layer with output stride of 16 and 32, respectively. The rest of SSDLite layers are attached on top of the last convolutional layer with output stride of 32. During training, we use a batch size of 24 and all the models are trained for 240,000 iterations. For more detailed settings, readers can refer to [25, 31].

In Table 8, we show the results when different backbone networks are used. Obviously, with the nearly the same number of parameters and computation, SSDLite with our backbone improves the one with MobileNetV2 by nearly 1%. This demonstrates that the proposed network has better transferable capability compared to MobileNetV2.

4.6 Improving Architecture Search as Super-operators

It has been verified in previous subsections that our proposed sandglass block is more effective than the inverted residual block in both the classification task and the object detection task. From a holistic perspective, we can also regard a residual block as a

Table 8. Detection results on the Pascal VOC 2007 test set. As can be seen, using the same SSDLite320 detector, replacing the MobileNetV2 backbone with our network achieves better results. Note that the multipliers of both MobileNetV2 and our network are set to 1.0.

No.	Method	Backbone	Param. (M)	M-Adds (B)	mAP (%)
1	SSD300	VGG [33]	36.1	35.2	77.2
2	SSDLite320	MobileNetV2 [31]	4.3	0.8	71.7
3	SSDLite320	MobileNeXt	4.3	0.8	72.6

Table 9. Results produced by different network architectures searched by DARTS [24]. For Lines 2 and 3, we separately add the inverted residual (IR) block and our sandglass block into the original search space of DARTS. We report results on CIFAR-10 dataset as in [24].

No.	Search Space	Test Error (%)	Param. (M)	Search Method	#Operators
1	DARTS original	3.11	3.25	gradient based	7
2	DARTS + IR Block	3.26	3.29	gradient based	8
3	DARTS + sandglass block	2.98	2.45	gradient based	8

'super' operator with more powerful transformation power than a regular convolutional operator. To further investigate the superiority of the proposed sandglass block over the inverted residual block, we separately add it into the search space of the differentiable searching algorithm (DARTS) [24] to see the network performance after architecture search and report the corresponding results on CIFAR-10 dataset. As shown in Table 9, by adding our sandglass block as an new operator into the DARTS search space without changing the cell structure, the resulting model achieves higher accuracy than the model with the original DARTS search space with about 25% parameter reduction. However, the searched model with the inverted residual block added in the search space decreases the original performance. This demonstrates that our proposed sandglass block can generate more expressive representations than the inverted residual block and can also be used in architecture search algorithms as a kind of 'super' operator. For more details on the searched cell structure, please refer to our supplementary materials.

5 Conclusions

In this paper, we deeply analyze the design rules and shortcomings of the previous inverted residual block. Based on the analysis, we propose to reverse the thought of adding shortcut connections between low-dimensional representations and present a novel building block, called the sandglass block, that connects high-dimensional representations instead. Experiments in both classification, object detection, and neural architecture search demonstrate the effectiveness of the proposed sandglass block and its potential to be used in more contexts.

Acknowledgement Jiashi Feng was partially supported by MOE Tier 2 MOE2017-T2-2-151, NUS_ECRA_FY17_P08, AISG-100E-2019-035.

References

1. Cai, H., Zhu, L., Han, S.: Proxylessnas: Direct neural architecture search on target task and hardware. arXiv preprint arXiv:1812.00332 (2018) 4, 13
2. Caron, M., Morcos, A., Bojanowski, P., Mairal, J., Joulin, A.: Pruning convolutional neural networks with self-supervision. arXiv preprint arXiv:2001.03554 (2020) 4
3. Chen, Y., Li, J., Xiao, H., Jin, X., Yan, S., Feng, J.: Dual path networks. In: Advances in neural information processing systems. pp. 4467–4475 (2017) 4
4. Chollet, F.: Xception: Deep learning with depthwise separable convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1251–1258 (2017) 5
5. Choukroun, Y., Kravchik, E., Yang, F., Kisilev, P.: Low-bit quantization of neural networks for efficient inference. arXiv preprint arXiv:1902.06822 (2019) 4
6. Dong, X., Yang, Y.: Nas-bench-102: Extending the scope of reproducible neural architecture search. arXiv preprint arXiv:2001.00326 (2020) 4
7. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html 9
8. Everingham, M., Eslami, S.A., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes challenge: A retrospective. International journal of computer vision **111**(1), 98–136 (2015) 12
9. Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single path one-shot neural architecture search with uniform sampling. arXiv preprint arXiv:1904.00420 (2019) 4
10. Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., Xu, C.: Ghostnet: More features from cheap operations. arXiv preprint arXiv:1911.11907 (2019) 13
11. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015) 4
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016) 1, 3, 4, 5, 8
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016) 3
14. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: European conference on computer vision. pp. 630–645. Springer (2016) 1
15. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015) 4
16. Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al.: Searching for mobilenetv3. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1314–1324 (2019) 4, 10, 11, 13
17. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017) 4, 5, 10, 13
18. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 7132–7141 (2018) 4, 13
19. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Quantized neural networks: Training neural networks with low precision weights and activations. The Journal of Machine Learning Research **18**(1), 6869–6898 (2017) 4
20. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. arXiv preprint arXiv:1405.3866 (2014) 4

21. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012) 9
22. Li, D., Zhou, A., Yao, A.: Hbonet: Harmonious bottleneck on two orthogonal dimensions. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 3316–3325 (2019) 4
23. Li, X., Wang, W., Hu, X., Yang, J.: Selective kernel networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 510–519 (2019) 4
24. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018) 3, 14
25. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: European conference on computer vision. pp. 21–37. Springer (2016) 3, 12, 13
26. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2736–2744 (2017) 4
27. Ma, N., Zhang, X., Zheng, H.T., Sun, J.: Shufflenet v2: Practical guidelines for efficient cnn architecture design. In: ECCV. pp. 116–131 (2018) 1, 4, 9
28. Migacz, S.: Nvidia 8-bit inference width tensorrt. In: GPU Technology Conference (2017) 10
29. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems. pp. 8024–8035 (2019) 9
30. Radu, V., Kaszyk, K., Wen, Y., Turner, J., Cano, J., Crowley, E.J., Franke, B., Storkey, A., O'Boyle, M.: Performance aware convolutional neural network channel pruning for embedded gpus (2019) 4
31. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: CVPR. pp. 4510–4520 (2018) 1, 2, 3, 4, 5, 7, 8, 10, 12, 13, 14
32. Sankararaman, K.A., De, S., Xu, Z., Huang, W.R., Goldstein, T.: The impact of neural network overparameterization on gradient confusion and stochastic gradient descent. arXiv preprint arXiv:1904.06963 (2019) 2, 5
33. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014) 14
34. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2820–2828 (2019) 1, 4, 13
35. Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: ICML (2019) 1, 4, 11, 12, 13
36. Tan, M., Le, Q.V.: Mixconv: Mixed depthwise convolutional kernels. CoRR, abs/1907.09595 (2019) 4
37. Touvron, H., Vedaldi, A., Douze, M., Jégou, H.: Fixing the train-test resolution discrepancy. In: Advances in Neural Information Processing Systems. pp. 8250–8260 (2019) 4
38. Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., Keutzer, K.: Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: CVPR. pp. 10734–10742 (2019) 13
39. Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1492–1500 (2017) 4

40. Ying, C., Klein, A., Real, E., Christiansen, E., Murphy, K., Hutter, F.: Nas-bench-101: Towards reproducible neural architecture search. arXiv preprint arXiv:1902.09635 (2019) 4
41. Zagoruyko, S., Komodakis, N.: Wide residual networks. arXiv preprint arXiv:1605.07146 (2016) 4
42. Zhou, D., Jin, X., Hou, Q., Wang, K., Yang, J., Feng, J.: Neural epitome search for architecture-agnostic network compression. In: International Conference on Learning Representations (2019) 4
43. Zhou, M., Liu, Y., Long, Z., Chen, L., Zhu, C.: Tensor rank learning in cp decomposition via convolutional neural network. Signal Processing: Image Communication **73**, 12–21 (2019) 4