ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices

Xiangyu Zhang*   Xinyu Zhou*   Mengxiao Lin   Jian Sun

Megvii Inc (Face++)

{zhangxiangyu,zxy,linmengxiao,sunjian}@megvii.com

Abstract

We introduce an extremely computation-efficient CNN architecture named ShuffleNet, which is designed specially for mobile devices with very limited computing power (e.g., 10-150 MFLOPs). The new architecture utilizes two new operations, pointwise group convolution and channel shuffle, to greatly reduce computation cost while maintaining accuracy. Experiments on ImageNet classification and MS COCO object detection demonstrate the superior performance of ShuffleNet over other structures, e.g. lower top-1 error (absolute 7.8%) than recent MobileNet [12] on ImageNet classification task, under the computation budget of 40 MFLOPs. On an ARM-based mobile device, ShuffleNet achieves ∼13× actual speedup over AlexNet while maintaining comparable accuracy.

1. Introduction

Building deeper and larger convolutional neural networks (CNNs) is a primary trend for solving major visual recognition tasks [22, 9, 34, 5, 29, 25]. The most accurate CNNs usually have hundreds of layers and thousands of channels [9, 35, 33, 41], thus requiring computation at billions of FLOPs. This report examines the opposite extreme: pursuing the best accuracy in very limited computational budgets at tens or hundreds of MFLOPs, focusing on common mobile platforms such as drones, robots, and smartphones. Note that many existing works [17, 23, 44, 43, 39, 28] focus on pruning, compressing, or low-bit representing a "basic" network architecture. Here we aim to explore a highly efficient basic architecture specially designed for our desired computing ranges.

We notice that state-of-the-art basic architectures such as *Xception* [3] and *ResNeXt* [41] become less efficient in extremely small networks because of the costly dense 1×1 convolutions. We propose using *pointwise group convolu-tions* to reduce computation complexity of 1×1 convolutions. To overcome the side effects brought by group convolutions, we come up with a novel *channel shuffle* operation to help the information flowing across feature channels. Based on the two techniques, we build a highly efficient architecture called *ShuffleNet*. Compared with popular structures like [31, 9, 41], for a given computation complexity budget, our ShuffleNet allows more feature map channels, which helps to encode more information and is especially critical to the performance of very small networks.

We evaluate our models on the challenging ImageNet classification [4, 30] and MS COCO object detection [24] tasks. A series of controlled experiments shows the effectiveness of our design principles and the better performance over other structures. Compared with the state-of-the-art architecture *MobileNet* [12], ShuffleNet achieves superior performance by a significant margin, e.g. absolute 7.8% lower ImageNet top-1 error at level of 40 MFLOPs.

We also examine the speedup on real hardware, i.e. an off-the-shelf ARM-based computing core. The ShuffleNet model achieves ∼**13**× *actual* speedup (theoretical speedup is 18×) over AlexNet [22] while maintaining comparable accuracy.

2. Related Work

Efficient Model Designs   The last few years have seen the success of deep neural networks in computer vision tasks [22, 37, 29], in which model designs play an important role. The increasing needs of running high quality deep neural networks on embedded devices encourage the study on efficient model designs [8]. For example, *GoogLeNet* [34] increases the depth of networks with much lower complexity compared to simply stacking convolution layers. *SqueezeNet* [14] reduces parameters and computation significantly while maintaining accuracy. *ResNet* [9, 10] utilizes the efficient bottleneck structure to achieve impressive performance. *SENet* [13] introduces an architectural unit that boosts performance at slight computation cost. Concurrent with us, a very re-
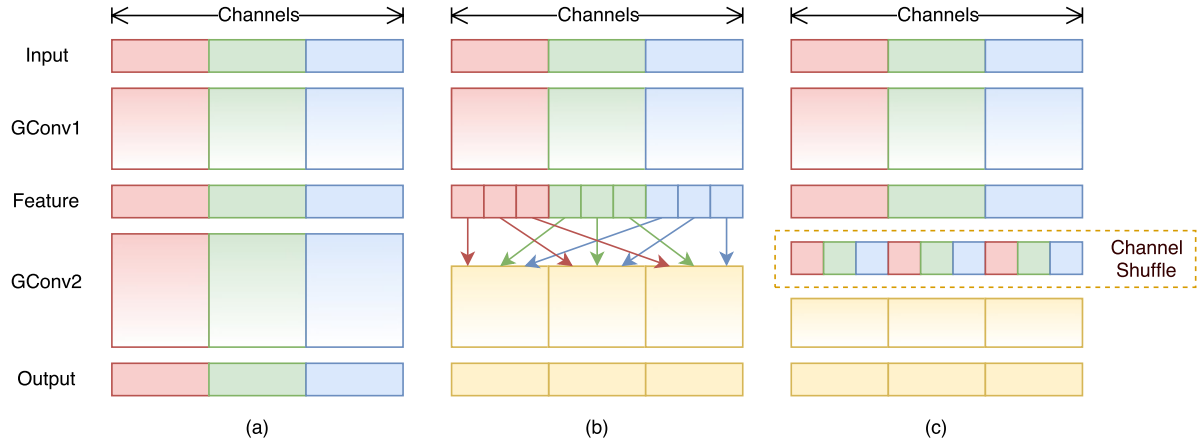
---

* Equal contribution.

Figure 1. Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when GConv2 takes data from different groups after GConv1; c) an equivalent implementation to b) using channel shuffle.

cent work [47] employs reinforcement learning and model search to explore efficient model designs. The proposed mobile *NASNet* model achieves comparable performance with our counterpart ShuffleNet model (26.0% @ 564 MFLOPs vs. 26.3% @ 524 MFLOPs for ImageNet classification error). But [47] do not report results on extremely tiny models (e.g. complexity less than 150 MFLOPs), nor evaluate the actual inference time on mobile devices.

Group Convolution The concept of group convolution, which was first introduced in *AlexNet* [22] for distributing the model over two GPUs, has been well demonstrated its effectiveness in ResNeXt [41] and DeepRoots [15]. Depthwise separable convolution proposed in Xception [3] generalizes the ideas of separable convolutions in Inception series [35, 33]. Recently, MobileNet [12] utilizes the depthwise separable convolutions and gains state-of-the-art results among lightweight models. Our work generalizes group convolution and depthwise separable convolution in a novel form.

Channel Shuffle Operation To the best of our knowledge, the idea of channel shuffle operation is rarely mentioned in previous work on efficient model design, although CNN library *cuda-convnet* [21] supports "random sparse convolution" layer, which is equivalent to random channel shuffle followed by a group convolutional layer. Such "random shuffle" operation has different purpose and been seldom exploited later. Very recently, another concurrent work [42] also adopt this idea for a two-stage convolution. However, [42] did not specially investigate the effectiveness of channel shuffle itself and its usage in tiny model design.

Model Acceleration This direction aims to accelerate inference while preserving accuracy of a pre-trained model. Pruning network connections [6, 7] or channels [39] reduces redundant connections in a pre-trained model while maintaining performance. Quantization [32, 28, 40, 46, 45] and factorization [23, 17, 19, 38] are proposed in literature to reduce redundancy in calculations to speed up inference. Without modifying the parameters, optimized convolution algorithms implemented by FFT [26, 36] and other methods [2] decrease time consumption in practice. Distilling [11] transfers knowledge from large models into small ones, which makes training small models easier.

3. Approach

3.1. Channel Shuffle for Group Convolutions

Modern convolutional neural networks [31, 34, 35, 33, 9, 10] usually consist of repeated building blocks with the same structure. Among them, state-of-the-art networks such as *Xception* [3] and *ResNeXt* [41] introduce efficient depthwise separable convolutions or group convolutions into the building blocks to strike an excellent trade-off between representation capability and computational cost. However, we notice that both designs do not fully take the 1×1 convolutions (also called *pointwise convolutions* in [12]) into account, which require considerable complexity. For example, in ResNeXt [41] only 3×3 layers are equipped with group convolutions. As a result, for each residual unit in ResNeXt the pointwise convolutions occupy 93.4% multiplication-adds (cardinality = 32 as suggested in [41]). In tiny networks, expensive pointwise convolutions result in limited number of channels to meet the complexity constraint, which might significantly damage the accuracy.

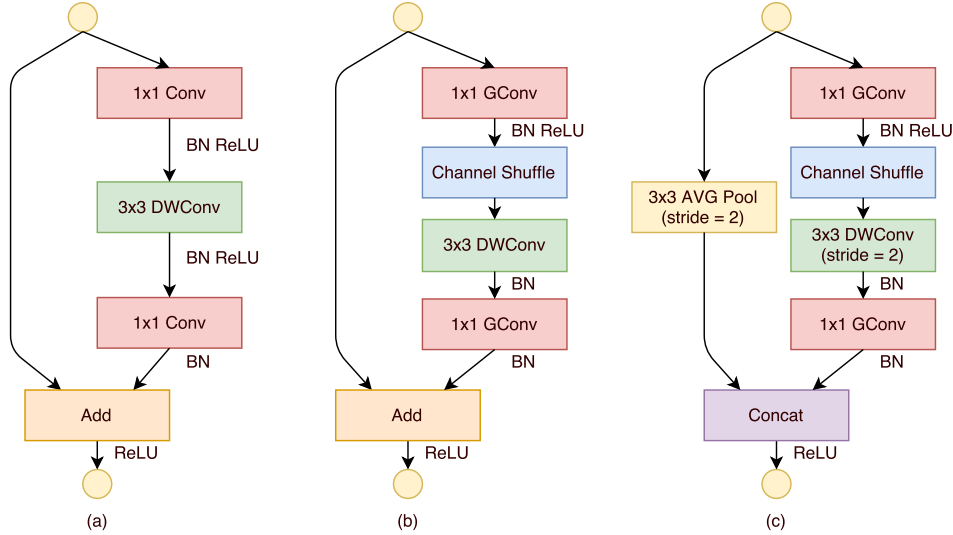To address the issue, a straightforward solution is to ap-

Figure 2. ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

ply channel sparse connections, for example group convolutions, also on 1×1 layers. By ensuring that each convolution operates only on the corresponding input channel group, group convolution significantly reduces computation cost. However, if multiple group convolutions stack together, there is one side effect: outputs from a certain channel are only derived from a small fraction of input channels. Fig 1 (a) illustrates a situation of two stacked group convolution layers. It is clear that outputs from a certain group only relate to the inputs within the group. This property blocks information flow between channel groups and weakens representation.

If we allow group convolution to obtain input data from different groups (as shown in Fig 1 (b)), the input and output channels will be fully related. Specifically, for the feature map generated from the previous group layer, we can first divide the channels in each group into several subgroups, then feed each group in the next layer with different subgroups. This can be efficiently and elegantly implemented by a *channel shuffle* operation (Fig 1 (c)): suppose a convolutional layer with g groups whose output has $g \times n$ channels; we first reshape the output channel dimension into (g, n), transposing and then flattening it back as the input of next layer. Note that the operation still takes effect even if the two convolutions have different numbers of groups. Moreover, channel shuffle is also differentiable, which means it can be embedded into network structures for end-to-end training.

Channel shuffle operation makes it possible to build more powerful structures with multiple group convolutional layers. In the next subsection we will introduce an efficient network unit with channel shuffle and group convolution.

3.2. ShuffleNet Unit

Taking advantage of the channel shuffle operation, we propose a novel *ShuffleNet* unit specially designed for small networks. We start from the design principle of bottleneck unit [9] in Fig 2 (a). It is a residual block. In its residual branch, for the 3×3 layer, we apply a computational economical 3×3 depthwise convolution [3] on the bottleneck feature map. Then, we replace the first 1×1 layer with pointwise group convolution followed by a channel shuffle operation, to form a ShuffleNet unit, as shown in Fig 2 (b). The purpose of the second pointwise group convolution is to recover the channel dimension to match the shortcut path. For simplicity, we do not apply an extra channel shuffle operation after the second pointwise layer as it results in comparable scores. The usage of batch normalization (BN) [16] and nonlinearity is similar to [9, 41], except that we do not use ReLU after depthwise convolution as suggested by [3]. As for the case where ShuffleNet is applied with stride, we simply make two modifications (see Fig 2 (c)): (i) add a 3×3 average pooling on the shortcut path; (ii) replace the element-wise addition with channel concatenation, which makes it easy to enlarge channel dimension with little extra computation cost.

Thanks to pointwise group convolution with channel shuffle, all components in ShuffleNet unit can be computed efficiently. Compared with ResNet [9] (bottleneck design) and ResNeXt [41], our structure has less complexity under the same settings. For example, given the input size $c \times h \times w$ and the bottleneck channels m, ResNet unit requires $hw(2cm + 9m^2)$ FLOPs and ResNeXt has $hw(2cm + 9m^2/g)$ FLOPs, while our ShuffleNet unit requires only $hw(2cm/g + 9m)$ FLOPs, where g means the

Layer	Output size	KSize	Stride	Repeat	Output channels (g groups)				
					$g=1$	$g=2$	$g=3$	$g=4$	$g=8$
Image	224×224				3	3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24	24
MaxPool	56×56	3×3	2						
Stage2	28×28		2	1	144	200	240	272	384
	28×28		1	3	144	200	240	272	384
Stage3	14×14		2	1	288	400	480	544	768
	14×14		1	7	288	400	480	544	768
Stage4	7×7		2	1	576	800	960	1088	1536
	7×7		1	3	576	800	960	1088	1536
GlobalPool	1×1	7×7							
FC					1000	1000	1000	1000	1000
Complexity					143M	140M	137M	133M	137M

Table 1. ShuffleNet architecture. The complexity is evaluated with FLOPs, i.e. the number of floating-point multiplication-adds. Note that for Stage 2, we do not apply group convolution on the first pointwise layer because the number of input channels is relatively small.

Model	Complexity (MFLOPs)	Classification error (%)				
		$g=1$	$g=2$	$g=3$	$g=4$	$g=8$
ShuffleNet 1$\times$	140	33.6	32.7	32.6	32.8	**32.4**
ShuffleNet 0.5$\times$	38	45.1	44.4	43.2	**41.6**	42.3
ShuffleNet 0.25$\times$	13	57.1	56.8	55.0	54.2	**52.7**

Table 2. Classification error vs. number of groups g (*smaller number represents better performance*)

number of groups for convolutions. In other words, given a computational budget, ShuffleNet can use wider feature maps. We find this is critical for small networks, as tiny networks usually have an insufficient number of channels to process the information.

In addition, in ShuffleNet depthwise convolution only performs on bottleneck feature maps. Even though depthwise convolution usually has very low theoretical complexity, we find it difficult to efficiently implement on low-power mobile devices, which may result from a worse computation/memory access ratio compared with other dense operations. Such drawback is also referred in [3], which has a runtime library based on TensorFlow [1]. In ShuffleNet units, we intentionally use depthwise convolution only on bottleneck in order to prevent overhead as much as possible.

3.3. Network Architecture

Built on ShuffleNet units, we present the overall ShuffleNet architecture in Table 1. The proposed network is mainly composed of a stack of ShuffleNet units grouped into three stages. The first building block in each stage is applied with stride = 2. Other hyper-parameters within a stage stay the same, and for the next stage the output channels are doubled. Similar to [9], we set the number of bottleneck channels to 1/4 of the output channels for each ShuffleNet

unit. Our intent is to provide a reference design as simple as possible, although we find that further hyper-parameter tunning might generate better results.

In ShuffleNet units, group number g controls the connection sparsity of pointwise convolutions. Table 1 explores different group numbers and we adapt the output channels to ensure overall computation cost roughly unchanged ($\sim$140 MFLOPs). Obviously, larger group numbers result in more output channels (thus more convolutional filters) for a given complexity constraint, which helps to encode more information, though it might also lead to degradation for an individual convolutional filter due to limited corresponding input channels. In Sec 4.1.1 we will study the impact of this number subject to different computational constrains.

To customize the network to a desired complexity, we can simply apply a scale factor s on the number of channels. For example, we denote the networks in Table 1 as "ShuffleNet 1$\times$", then "ShuffleNet $s\times$" means scaling the number of filters in ShuffleNet 1$\times$ by s times thus overall complexity will be roughly s^2 times of ShuffleNet 1$\times$.

4. Experiments

We mainly evaluate our models on the ImageNet 2012 classification dataset [30, 4]. We follow most of the training settings and hyper-parameters used in [41], with two exceptions: (i) we set the weight decay to 4e-5 instead of

Model	Cls err. (%, no shuffle)	Cls err. (%, shuffle)	Δ err. (%)
ShuffleNet 1x ($g = 3$)	34.5	**32.6**	1.9
ShuffleNet 1x ($g = 8$)	37.6	**32.4**	5.2
ShuffleNet 0.5x ($g = 3$)	45.7	**43.2**	2.5
ShuffleNet 0.5x ($g = 8$)	48.1	**42.3**	5.8
ShuffleNet 0.25x ($g = 3$)	56.3	**55.0**	1.3
ShuffleNet 0.25x ($g = 8$)	56.5	**52.7**	3.8

Table 3. ShuffleNet with/without channel shuffle (*smaller number represents better performance*)

1e-4 and use linear-decay learning rate policy (decreased from 0.5 to 0); (ii) we use slightly less aggressive scale augmentation for data preprocessing. Similar modifications are also referenced in [12] because such small networks usually suffer from underfitting rather than overfitting. It takes 1 or 2 days to train a model for 3×10^5 iterations on 4 GPUs, whose batch size is set to 1024. To benchmark, we compare single crop top-1 performance on ImageNet validation set, i.e. cropping 224×224 center view from $256 \times$ input image and evaluating classification accuracy. We use exactly the same settings for all models to ensure fair comparisons.

4.1. Ablation Study

The core idea of ShuffleNet lies in pointwise group convolution and channel shuffle operation. In this subsection we evaluate them respectively.

4.1.1 Pointwise Group Convolutions

To evaluate the importance of pointwise group convolutions, we compare ShuffleNet models of the same complexity whose numbers of groups range from 1 to 8. If the group number equals 1, no pointwise group convolution is involved and then the ShuffleNet unit becomes an "Xception-like" [3] structure. For better understanding, we also scale the width of the networks to 3 different complexities and compare their classification performance respectively. Results are shown in Table 2.

From the results, we see that models with group convolutions ($g > 1$) consistently perform better than the counterparts without pointwise group convolutions ($g = 1$). Smaller models tend to benefit more from groups. For example, for ShuffleNet $1\times$ the best entry ($g = 8$) is 1.2% better than the counterpart, while for ShuffleNet $0.5\times$ and $0.25\times$ the gaps become 3.5% and 4.4% respectively. Note that group convolution allows more feature map channels for a given complexity constraint, so we hypothesize that the performance gain comes from wider feature maps which help to encode more information. In addition, a smaller network involves thinner feature maps, meaning it benefits more from enlarged feature maps.

Table 2 also shows that for some models (e.g. ShuffleNet $0.5\times$) when group numbers become relatively large

(e.g. $g = 8$), the classification score saturates or even drops. With an increase in group number (thus wider feature maps), input channels for each convolutional filter become fewer, which may harm representation capability. Interestingly, we also notice that for smaller models such as ShuffleNet $0.25\times$ larger group numbers tend to better results consistently, which suggests wider feature maps bring more benefits for smaller models.

4.1.2 Channel Shuffle vs. No Shuffle

The purpose of shuffle operation is to enable cross-group information flow for multiple group convolution layers. Table 3 compares the performance of ShuffleNet structures (group number is set to 3 or 8 for instance) with/without channel shuffle. The evaluations are performed under three different scales of complexity. It is clear that channel shuffle consistently boosts classification scores for different settings. Especially, when group number is relatively large (e.g. $g = 8$), models with channel shuffle outperform the counterparts by a significant margin, which shows the importance of cross-group information interchange.

4.2. Comparison with Other Structure Units

Recent leading convolutional units in VGG [31], ResNet [9], GoogleNet [34], ResNeXt [41] and Xception [3] have pursued state-of-the-art results with large models (e.g. $\geq$ 1GFLOPs), but do not fully explore low-complexity conditions. In this section we survey a variety of building blocks and make comparisons with ShuffleNet under the same complexity constraint.

For fair comparison, we use the overall network architecture as shown in Table 1. We replace the ShuffleNet units in Stage 2-4 with other structures, then adapt the number of channels to ensure the complexity remains unchanged. The structures we explored include:

- *VGG-like*. Following the design principle of VGG net [31], we use a two-layer 3×3 convolutions as the basic building block. Different from [31], we add a Batch Normalization layer [16] after each of the convolutions to make end-to-end training easier.

Complexity (MFLOPs)	VGG-like	ResNet	Xception-like	ResNeXt	ShuffleNet (ours)
140	50.7	37.3	33.6	33.3	**32.4** ($1\times$, $g=8$)
38	-	48.8	45.1	46.0	**41.6** ($0.5\times$, $g=4$)
13	-	63.7	57.1	65.2	**52.7** ($0.25\times$, $g=8$)

Table 4. Classification error vs. various structures (%, *smaller number represents better performance*). We do not report VGG-like structure on smaller networks because the accuracy is significantly worse.

Model	Complexity (MFLOPs)	Cls err. (%)	Δ err. (%)
1.0 MobileNet-224	569	29.4	-
ShuffleNet $2\times$ ($g=3$)	524	**26.3**	3.1
ShuffleNet $2\times$ (with *SE*[13], $g=3$)	527	**24.7**	4.7
0.75 MobileNet-224	325	31.6	-
ShuffleNet $1.5\times$ ($g=3$)	292	**28.5**	3.1
0.5 MobileNet-224	149	36.3	-
ShuffleNet $1\times$ ($g=8$)	140	**32.4**	3.9
0.25 MobileNet-224	41	49.4	-
ShuffleNet $0.5\times$ ($g=4$)	38	**41.6**	7.8
ShuffleNet $0.5\times$ (shallow, $g=3$)	40	42.8	6.6

Table 5. ShuffleNet vs. MobileNet [12] on ImageNet Classification

- *ResNet*. We adopt the "bottleneck" design in our experiment, which has been demonstrated more efficient in [9] . Same as [9], the *bottleneck ratio*[1] is also $1:4$.

- *Xception-like*. The original structure proposed in [3] involves fancy designs or hyper-parameters for different stages, which we find difficult for fair comparison on small models. Instead, we remove the pointwise group convolutions and channel shuffle operation from ShuffleNet (also equivalent to ShuffleNet with $g=1$). The derived structure shares the same idea of "depthwise separable convolution" as in [3], which is called an *Xception-like* structure here.

- *ResNeXt*. We use the settings of *cardinality* $=16$ and bottleneck ratio $=1:2$ as suggested in [41]. We also explore other settings, e.g. bottleneck ratio $=1:4$, and get similar results.

We use exactly the same settings to train these models. Results are shown in Table 4. Our ShuffleNet models outperform most others by a significant margin under different complexities. Interestingly, we find an empirical relationship between feature map channels and classification accuracy. For example, under the complexity of 38 MFLOPs, output channels of Stage 4 (see Table 1) for VGG-like, ResNet, ResNeXt, Xception-like, ShuffleNet models are 50, 192, 192, 288, 576 respectively, which is consistent with

---

[1]In the bottleneck-like units (like ResNet, ResNeXt or ShuffleNet) *bottleneck ratio* implies the ratio of bottleneck channels to output channels. For example, bottleneck ratio = 1 : 4 means the output feature map is 4 times the width of the bottleneck feature map.

the increase of accuracy. Since the efficient design of ShuffleNet, we can use more channels for a given computation budget, thus usually resulting in better performance.

Note that the above comparisons do not include *GoogleNet* or *Inception* series [34, 35, 33]. We find it nontrivial to generate such Inception structures to small networks because the original design of Inception module involves too many hyper-parameters. As a reference, the first GoogleNet version [34] has 31.3% top-1 error at the cost of 1.5 GFLOPs (See Table 6). More sophisticated Inception versions [35, 33] are more accurate, however, involve significantly increased complexity. Recently, Kim et al. propose a lightweight network structure named *PVANET* [20] which adopts Inception units. Our reimplemented PVANET (with 224×224 input size) has 29.7% classification error with a computation complexity of 557 MFLOPs, while our ShuffleNet 2x model ($g=3$) gets 26.3% with 524 MFLOPs (see Table 6).

4.3. Comparison with MobileNets and Other Frameworks

Recently Howard et al. have proposed *MobileNets* [12] which mainly focus on efficient network architecture for mobile devices. MobileNet takes the idea of depthwise separable convolution from [3] and achieves state-of-the-art results on small models.

Table 5 compares classification scores under a variety of complexity levels. It is clear that our ShuffleNet models are superior to MobileNet for all the complexities. Though our ShuffleNet network is specially designed for small models ($<$ 150 MFLOPs), we find it is still better than MobileNet

Model	Cls err. (%)	Complexity (MFLOPs)
VGG-16 [31]	28.5	15300
ShuffleNet 2× ($g = 3$)	26.3	**524**
GoogleNet [34]*	31.3	1500
ShuffleNet 1× ($g = 8$)	32.4	**140**
AlexNet [22]	42.8	720
SqueezeNet [14]	42.5	833
ShuffleNet 0.5× ($g = 4$)	41.6	**38**

Table 6. Complexity comparison. *Implemented by BVLC (https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet)*

Model	mAP [.5, .95] (300× image)	mAP [.5, .95] (600× image)
ShuffleNet 2× ($g = 3$)	**18.7%**	**25.0%**
ShuffleNet 1× ($g = 3$)	14.5%	19.8%
1.0 MobileNet-224 [12]	16.4%	19.8%
1.0 MobileNet-224 (our impl.)	14.9%	19.3%

Table 7. Object detection results on MS COCO (*larger numbers represents better performance*). For MobileNets we compare two results: 1) COCO detection scores reported by [12]; 2) finetuning from our reimplemented MobileNets, whose training and finetuning settings are exactly the same as that for ShuffleNets.

Model	Cls err. (%)	FLOPs	224×224	480×640	720×1280
ShuffleNet 0.5× ($g = 3$)	43.2	38M	15.2ms	87.4ms	260.1ms
ShuffleNet 1× ($g = 3$)	32.6	140M	37.8ms	222.2ms	684.5ms
ShuffleNet 2× ($g = 3$)	26.3	524M	108.8ms	617.0ms	1857.6ms
AlexNet [22]	42.8	720M	184.0ms	1156.7ms	3633.9ms
1.0 MobileNet-224 [12]	29.4	569M	110.0ms	612.0ms	1879.2ms

Table 8. Actual inference time on mobile device (*smaller number represents better performance*). The platform is based on a single Qualcomm Snapdragon 820 processor. All results are evaluated with **single thread**.

for higher computation cost, e.g. 3.1% more accurate than MobileNet 1× at the cost of 500 MFLOPs. For smaller networks (∼40 MFLOPs) ShuffleNet surpasses MobileNet by **7.8%**. Note that our ShuffleNet architecture contains 50 layers while MobileNet only has 28 layers. For better understanding, we also try ShuffleNet on a 26-layer architecture by removing half of the blocks in Stage 2-4 (see "ShuffleNet 0.5× shallow ($g = 3$)" in Table 5). Results show that the shallower model is still significantly better than the corresponding MobileNet, which implies that the effectiveness of ShuffleNet mainly results from its efficient structure, not the depth.

Table 6 compares our ShuffleNet with a few popular models. Results show that with similar accuracy ShuffleNet is much more efficient than others. For example, ShuffleNet 0.5× is theoretically 18× faster than AlexNet [22] with comparable classification score. We will evaluate the actual running time in Sec 4.5.

It is also worth noting that the simple architecture design makes it easy to equip ShuffeNets with the latest advances such as [13, 27]. For example, in [13] the authors propose *Squeeze-and-Excitation (SE)* blocks which achieve

state-of-the-art results on large ImageNet models. We find SE modules also take effect in combination with the backbone ShuffleNets, for instance, boosting the top-1 error of ShuffleNet 2× to 24.7% (shown in Table 5). Interestingly, though negligible increase of theoretical complexity, we find ShuffleNets with SE modules are usually $25 \sim 40\%$ slower than the "raw" ShuffleNets on mobile devices, which implies that actual speedup evaluation is critical on low-cost architecture design. In Sec 4.5 we will make further discussion.

4.4. Generalization Ability

To evaluate the generalization ability for transfer learning, we test our ShuffleNet model on the task of MS COCO object detection [24]. We adopt Faster-RCNN [29] as the detection framework and use the publicly released Caffe code [29, 18] for training with default settings. Similar to [12], the models are trained on the COCO train+val dataset excluding 5000 minival images and we conduct testing on the minival set. Table 7 shows the comparison of results trained and evaluated on two input resolutions. Comparing ShuffleNet 2× with MobileNet whose complexity are com-

parable (524 vs. 569 MFLOPs), our ShuffleNet 2× surpasses MobileNet by a significant margin on both resolutions; our ShuffleNet 1× also achieves comparable results with MobileNet on 600× resolution, but has ∼4× complexity reduction. We conjecture that this significant gain is partly due to ShuffleNet's simple design of architecture without bells and whistles.

4.5. Actual Speedup Evaluation

Finally, we evaluate the actual inference speed of ShuffleNet models on a mobile device with an ARM platform. Though ShuffleNets with larger group numbers (e.g. $g = 4$ or $g = 8$) usually have better performance, we find it less efficient in our current implementation. Empirically $g = 3$ usually has a proper trade-off between accuracy and actual inference time. As shown in Table 8, three input resolutions are exploited for the test. Due to memory access and other overheads, we find every 4× theoretical complexity reduction usually results in ∼2.6× actual speedup in our implementation. Nevertheless, compared with AlexNet [22] our ShuffleNet 0.5× model still achieves ∼13× **actual** speedup under comparable classification accuracy (the theoretical speedup is 18×), which is much faster than previous AlexNet-level models or speedup approaches such as [14, 17, 23, 43, 44, 39].

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. 4

[2] H. Bagherinezhad, M. Rastegari, and A. Farhadi. Lcnn: Lookup-based convolutional neural network. *arXiv preprint arXiv:1611.06473*, 2016. 2

[3] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016. 1, 2, 3, 4, 5, 6

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009. 1, 4

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014. 1

[6] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 2

[7] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in*

[8] K. He and J. Sun. Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5353–5360, 2015. 1

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 1, 2, 3, 4, 5, 6

[10] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016. 1, 2

[11] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1, 2, 3, 5, 6, 7

[13] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017. 1, 6, 7

[14] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 1, 7, 8

[15] Y. Ioannou, D. Robertson, R. Cipolla, and A. Criminisi. Deep roots: Improving cnn efficiency with hierarchical filter groups. *arXiv preprint arXiv:1605.06489*, 2016. 2

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 3, 5

[17] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014. 1, 2, 8

[18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014. 7

[19] J. Jin, A. Dundar, and E. Culurciello. Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474*, 2014. 2

[20] K.-H. Kim, S. Hong, B. Roh, Y. Cheon, and M. Park. Pvanet: Deep but lightweight neural networks for real-time object detection. *arXiv preprint arXiv:1608.08021*, 2016. 6

[21] A. Krizhevsky. cuda-convnet: High-performance c++/cuda implementation of convolutional neural networks, 2012. 2

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1, 2, 7, 8

[23] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014. 1, 2, 8

[24] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014. 1, 7

[25] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. 1

[26] M. Mathieu, M. Henaff, and Y. LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013. 2

[27] P. Ramachandran, B. Zoph, and Q. V. Le. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941*, 2017. 7

[28] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnornet: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016. 1, 2

[29] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 1, 7

[30] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 1, 4

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 2, 5, 7

[32] D. Soudry, I. Hubara, and R. Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems*, pages 963–971, 2014. 2

[33] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016. 1, 2, 6

[34] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015. 1, 2, 5, 6, 7

[35] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016. 1, 2, 6

[36] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun. Fast convolutional nets with fbfft: A gpu performance evaluation. *arXiv preprint arXiv:1412.7580*, 2014. 2

[37] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015. 1

[38] M. Wang, B. Liu, and H. Foroosh. Design of efficient convolutional layers using single intra-channel convolution, topological subdivisioning and spatial "bottleneck" structure. *arXiv preprint arXiv:1608.04337*, 2016. 2

[39] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016. 1, 2, 8

[40] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016. 2

[41] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016. 1, 2, 3, 4, 5, 6

[42] T. Zhang, G.-J. Qi, B. Xiao, and J. Wang. Interleaved group convolutions for deep neural networks. In *International Conference on Computer Vision*, 2017. 2

[43] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2016. 1, 8

[44] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1984–1992, 2015. 1, 8

[45] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017. 2

[46] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016. 2

[47] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017. 2