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, designed specially for mobile devices with very limited computing power (e.g., 10-150 MFLOPs). The new architecture utilizes two proposed 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 6.7%) than the recent MobileNet system on ImageNet classification under the computation budget of 40 MFLOPs. On an ARM-based mobile device, ShuffleNet achieves ~13$\times$ actual speedup over AlexNet while maintaining comparable accuracy.

1 Introduction

Building deeper and larger convolutional neural networks (CNNs) is a primary trend in the development of major visual tasks [19, 9, 30, 5, 25, 22]. The most accurate CNNs usually have hundreds of layers and thousands of channels [9, 31, 29, 37], 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 phones. Note that many existing works [15, 20, 39, 38, 35, 24] 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* [37] become less efficient in extremely small networks because of the costly dense 1×1 convolutions. We propose using *pointwise group convolutions* instead to reduce computation complexity. To overcome the side effects brought by pointwise 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 [27, 9, 37], 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, 26] and MS COCO object detection [21] 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 6.7% lower ImageNet top-1 error at level of 40 MFLOPs.

*Equally contribution.

We also examine the speedup on real hardware, i.e. an off-the-shelf ARM-based computing core. The ShuffleNet model achieves **~13**$\times$ *actual* speedup (theoretical speedup is 18$\times$) over AlexNet [19] 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[19, 33, 25], 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[30] increases the depth of networks with much lower complexity compared to simply stacking convolution layers. SqueezeNet[13] reduces parameters and computation significantly while maintaining accuracy. ResNet[9, 10] utilizes the efficient bottleneck structure to achieve impressive performance. The concept of group convolution, which should be first introduced in AlexNet[19] for distributing the model over two GPUs, has shown the potential to improve accuracy in [37]. Depthwise separable convolution proposed in Xception[3] generalizes the ideas of separable convolutions in Inception series[31, 29]. 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.

Model Acceleration This direction aims to accelerate inference while preserving accuracy of a pretrained model. Pruning network connections[6, 7] or channels[35] reduces redundant connections in a pre-trained model while maintaining performance. Quantization[28, 24, 36, 40] and factorization[20, 15, 17, 34] are proposed in literature to reduce redundancy in calculations to speed up inference. Without modifying the parameters, optimized convolution algorithms implemented by FFT[23, 32] 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. Compared to these methods, our approach focuses on better model designs to improve performance rather than accelerating or transferring an existing model.

3 Approach

3.1 Channel Shuffle for Group Convolutions

Modern convolutional neural networks [27, 30, 31, 29, 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* [37] 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 [37] 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 [37]). 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 apply 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

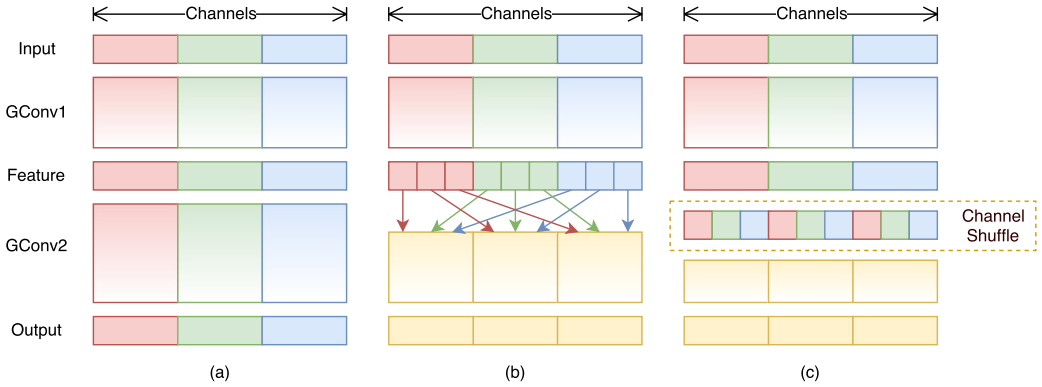

Figure 1: Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when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

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 convolutions.

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 layers 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) [14] and nonlinearity is similar to [9, 37], 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 [37], 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 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 pass 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.

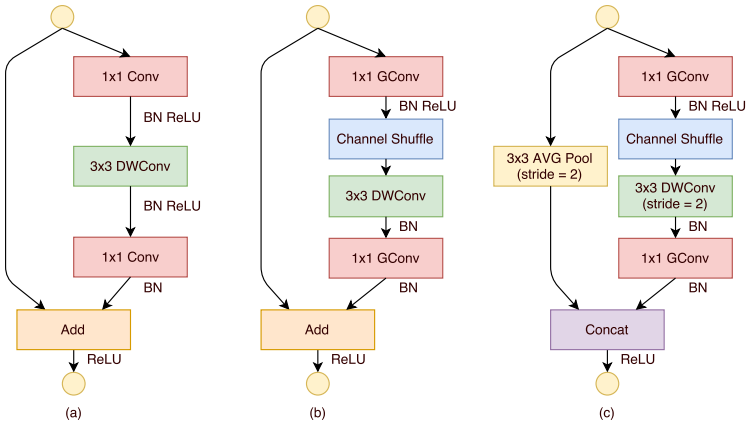

Figure 2: ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

Table 1: ShuffleNet architecture

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[1]	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[2]					143M	140M	137M	133M	137M

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 (~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

[1]We do not apply group convolution on the first pointwise layer because the number of input channels is relatively small.

[2]Evaluated with FLOPs, i.e. the number of floating-point multiplication-adds.

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

Model	Complexity (MFLOPs)	Classification error (%)				
		$g = 1$	$g = 2$	$g = 3$	$g = 4$	$g = 8$
ShuffleNet $1\times$	140	35.1	34.2	**34.1**	34.3	34.7
ShuffleNet $0.5\times$	38	46.1	45.1	44.4	**43.7**	43.8
ShuffleNet $0.25\times$	13	56.7	56.3	55.6	54.5	**53.7**
ShuffleNet $0.5\times$ (arch2)	40	45.7	44.3	43.8	43.2	**42.7**
ShuffleNet $0.25\times$ (arch2)	13	56.5	55.3	55.5	54.3	**53.3**

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 2016 classification dataset [26, 4]. We use our self-developed system for training, and follow most of the training settings and hyper-parameters used in [37], with two exceptions: (i) we set the weight decay to 4e-5 instead of 1e-4; (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. 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.

4.1 Ablation Study

The core idea of ShuffleNet lies in pointwise group convolution and channel shuffle operation referenced in Sections 3.1 and 3.2. In this subsection we evaluate them respectively.

4.1.1 On the Importance of 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 ($g = 3$) is 1.0% better than the counterpart, while for ShuffleNet $0.5\times$ and $0.25\times$ the gaps become 2.4% and 3.0% 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 $1\times$) when group numbers become relatively large (e.g. $g = 4$ or 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 are especially important for smaller models. Inspired by this, for low-cost models we slightly modify the architecture in Table 1: we remove two units in Stage3 and widen each feature map to maintain the overall complexity. Results of the new architecture (named "*arch2*") are shown

in Table 2. It is clear that the newly designed models are consistently better than the counterparts; besides, pointwise group convolution still takes effect in the architecture.

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.

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

Model	Cls err. (%, no shuffle)	Cls err. (%, shuffle)	Δ err. (%)
ShuffleNet 1x ($g = 3$)	36.4	**34.1**	2.3
ShuffleNet 0.5x ($g = 3$)	46.1	**44.4**	1.7
ShuffleNet 0.25x ($g = 3$)	56.5	**55.6**	0.9
ShuffleNet 0.25x (arch2, $g = 3$)	56.6	**55.5**	1.1
ShuffleNet 0.5x (arch2, $g = 8$)	46.2	**42.7**	3.5
ShuffleNet 0.25x (arch2, $g = 8$)	57.3	**53.3**	4.0

4.2 Comparison with Other Structure Units

Recent leading convolutional units in VGG [27], ResNet [9], GoogleNet [30], ResNeXt [37] 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 [27], we use a two-layer 3×3 convolutions as the basic building block. Different from [27], we add a Batch Normalization layer [14] after each of the convolutions to make end-to-end training easier.
- *ResNet*. We adopt the "bottleneck" design in our experiment, which has been demonstrated more efficient in [9]. Same as [9], the *bottleneck ratio*[3] 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 [37]. 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

[3]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.

Table 4: Classification error vs. various structures (%, *smaller number represents better performance*)

Complexity (MFLOPs)	VGG-like[4]	ResNet	Xception-like	ResNeXt	ShuffleNet (ours)
140	56.0	38.7	35.1	34.3	**34.1** ($1\times$, $g = 3$)
38	-	48.9	46.1	46.3	**43.7** ($0.5\times$, $g = 4$)
13	-	61.6	56.7	59.2	**53.7** ($0.25\times$, $g = 8$)
40 (arch2)	-	48.5	45.7	47.2	**42.7** ($0.5\times$, $g = 8$)
13 (arch2)	-	61.3	56.5	61.0	**53.3** ($0.25\times$, $g = 8$)

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

Model	Complexity (MFLOPs)	Cls err. (%)	Δ err. (%)
1.0 MobileNet-224	569	29.4	-
ShuffleNet $2\times$ ($g = 3$)	524	**29.1**	0.3
0.75 MobileNet-224	325	31.6	-
ShuffleNet $1.5\times$ ($g = 3$)	292	**31.0**	0.6
0.5 MobileNet-224	149	36.3	-
ShuffleNet $1\times$ ($g = 3$)	140	**34.1**	2.2
0.25 MobileNet-224	41	49.4	-
ShuffleNet $0.5\times$ (arch2, $g = 8$)	40	**42.7**	6.7
ShuffleNet $0.5\times$ (shallow, $g = 3$)	40	45.2	4.2

consistent with 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.

Table 4 also evaluates various models on a shallower but wider architecture (marked as "*arch2*", see 4.1.1 for details). It can be observed that ShuffleNet still achieves outstanding results over other structures.

Note that the above comparisons do not include GoogleNet or Inception series [30, 31, 29]. 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, Kim et al. have recently proposed a lightweight network structure named *PVANET* [18] which adopts Inception units. Our reimplemented PVANET (with 224×224 input size) has 35.3% classification error with a computation complexity of 557 MFLOPs, while our ShuffleNet 2x model ($g = 3$) gets 29.1% with 524 MFLOPs (see Table 6). So even though lack of direct comparisons, it is unlikely for Inception to surpass ShuffleNet in trivial settings.

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 slightly better than MobileNet for higher computation cost. For smaller networks (~40 MFLOPs) ShuffleNet surpasses MobileNet by **6.7%**. Note that our ShuffleNet architecture contains 50 layers (or 44 layers for *arch2*) 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 $0.5\times$ 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.

[4]We do not report VGG-like structure on smaller networks because the accuracy is significantly worse.

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 [19] with comparable classification score. We will evaluate the actual running time in Sec 4.5.

Table 6: Complexity comparison

Model	Cls err. (%)	Complexity (MFLOPs)
VGG-16 [27]	28.5	15300
ShuffleNet 2× ($g = 3$)	29.1	**524**
PVANET [18] (*our impl.*)	35.3	557
ShuffleNet 1× ($g = 3$)	34.1	**140**
AlexNet [19]	42.8	720
SqueezeNet [13]	42.5	833
ShuffleNet 0.5× (arch2, $g = 8$)	42.7	**40**

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 [21]. We adopt Faster-RCNN [25] as the detection framework and use the publicly released Caffe code [25, 16] 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 comparable (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.

Table 7: Object detection results on MS COCO (*larger numbers represents better performance*)

Model	mAP $[.5, .95]$ (300× image)	mAP $[.5, .95]$ (600× image)
ShuffleNet 2× ($g = 3$)	**18.0%**	**24.5%**
ShuffleNet 1× ($g = 3$)	14.3%	19.8%
1.0 MobileNet-224 [12]	16.4%	19.8%

4.5 Actual Speedup Evaluation

Finally, we evaluate the actual inference speed of ShuffleNet models on a mobile device with an ARM platform. 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 [19] 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 [13, 15, 20, 38, 39, 35].

Table 8: Actual inference time on mobile device (*smaller number represents better performance*)

Model	Cls err. (%)	FLOPs	224 × 224	480 × 640	720 × 1280
ShuffleNet 0.5× (arch2, $g = 3$)	43.8	40M	15.2ms	87.4ms	260.1ms
ShuffleNet 1× ($g = 3$)	34.1	140M	37.8ms	222.2ms	684.5ms
AlexNet [19]	42.8	720M	184.0ms	1156.7ms	3633.9ms

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. Lcnn: Lookup-based convolutional neural network. *arXiv preprint arXiv:1611.06473*, 2016.

[3] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li 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.

[5] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra 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.

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

[7] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

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

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[11] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[12] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[13] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[15] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[16] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[17] Jonghoon Jin, Aysegul Dundar, and Eugenio Culurciello. Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474*, 2014.

[18] Kye-Hyeon Kim, Sanghoon Hong, Byungseok Roh, Yeongjae Cheon, and Minje Park. Pvanet: Deep but lightweight neural networks for real-time object detection. *arXiv preprint arXiv:1608.08021*, 2016.

[19] 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.

[20] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

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

[22] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[23] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

[24] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[25] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

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

[28] Daniel Soudry, Itay Hubara, and Ron 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.

[29] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[30] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[31] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[32] Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A gpu performance evaluation. *arXiv preprint arXiv:1412.7580*, 2014.

[33] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru 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.

[34] Min Wang, Baoyuan Liu, and Hassan Foroosh. Design of efficient convolutional layers using single intra-channel convolution, topological subdivisioning and spatial "bottleneck" structure. *arXiv preprint arXiv:1608.04337*, 2016.

[35] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[36] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[37] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016.

[38] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2016.

[39] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian 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.

[40] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.