

**AugShuffleNet: Communicate More, Compute Less**

LongQing Ye

**Abstract**

As a remarkable compact model, ShuffleNetV2 offers a good example to design efficient ConvNets but its limit is rarely noticed. In this paper, we rethink the design pattern of ShuffleNetV2 and find that the channel-wise redundancy problem still constrains the efficiency improvement of Shuffle block in the wider ShuffleNetV2. To resolve this issue, we propose another augmented variant of shuffle block in the form of bottleneck-like structure and more implicit short connections. To verify the effectiveness of this building block, we further build a more powerful and efficient model family, termed as AugShuffleNets. Evaluated on the CIFAR-10 and CIFAR-100 datasets, AugShuffleNet consistently outperforms ShuffleNetV2 in terms of accuracy with less computational cost and fewer parameter count.

**1 Introduction**

Since the success of AlexNet [1], convolutional neural networks (CNNs, or ConvNets) have become dominant in various vision tasks including image classification, object detection and semantic segmentation. Typical models including VGG, GoogLeNet and Resnet [2–4] achieved remarkable performance. However, the accuracy improvement usually involves scaling up network in the form of more layers or more channels per layer, further leading to an increasing amount of computational cost and parameters. In recent years, many real-time CNN-based applications are deployed on resource-constrained platforms such as sensors and smart phones, requiring CNN models to be computationally efficient and of rapid response. Therefore, how to design efficient models has become an important research topic.

Many efforts has been dedicated to improve the efficiency of ConvNets. Currently, there are two main representative and complementary schemes: model compression and compact model design. The former kind aims to reduce the redundancy of a larger model without significant degeneration in performance via network pruning, quantization, low-rank, etc. However, the upper limit of model performance tends to be determined by the pre-trained network. The latter kind is trying to build compact models trained from scratch following . Typical instances like MobileNetV1-V3 and ShuffleNetV1-V2 utilize efficient operators, building block or network architecture search algorithms, providing significant insights for compact model design.

In this paper, we revisit the design of building block in ShuffleNetV2 and design a more efficient and powerful CNN model named AugShuffleNet.

**2 Related Work**

2.1 Compact Model Design

Composition of low-rank $(1 \times 3, 3 \times 1)$ filters [5] or sparse filters [6] are proposed to approximate dense convolution filter. Those work indicates that less redundant filters can bring a great reduction of FLOPs and parameters while maintaining the performance of models. Group convolution has been viewed as a standard operator in modern compact models [6,7]. Depth-wise convolution is an extreme form of group convolution, in which each channel presents a group. ShuffleNetV1 [6] uses group convolution to replace $1 \times 1$ convolution and further introduces the operation of "Channel Shuffle" to improve cross-group information communication. MobilenetV1 [8] utilizes depth-wise convolution and pointwise convolution to construct a lightweight model for mobile platforms. ResNet [4] and MobileNetV2 [9] adopt a bottleneck structure to alleviate the burden of heavy computation for channel expansion. MobileNetV3 and other
work\[10\]-[12]\ introduce the neural architecture search algorithms into compact model design, significantly reducing the cost of manual design.

2.2 Model Compression

Complementary to compact model design, model compression is another approach to further shrink pre-trained models. Network pruning\[13\] removes redundant and non-informative connections or channels. Model quantization\[14\] aims to represent stored weights at a low cost for model compression and calculation acceleration. Knowledge distillation\[15\] transfers refined knowledge from "teacher network" into "student network", simplifying the process of suppressing redundant information. In addition, efficient convolution algorithms like FFT\[16\] and winograd\[17\] are explored to speed up the implementation of convolutional layer without any modification of network design.

2.3 Short Connection

ResNets and Highway Networks\[18\] introduce skip connections to allow training deeper neural networks. Stochastic depth\[19\] reduces training depth of ResNets by randomly skipping layers. SkipNet further utilizes gating mechanism to learn how to skip layers for both training and inference procedures. Nowadays residual connection proposed by ResNet has been a popular technique to construct deep neural networks\[7,20\] in vision and natural language processing domain. Research\[21\] regards ResNet as a collection of many paths of different length and discovers that deep ResNet updates weights mainly via short paths during the training procedure. DenseNet\[22\] introduces densely connections by connecting each layer to every other layer in the networks to increase more short path. In general, under the framework of backpropagation neural network, short connections create more shorter paths to allow cross-layer gradient flow for smoother optimization while long paths in deep neural networks enable more complex information extraction and larger capacity to store temporary information during the dynamics process of training.

3 Approach

![Figure 1: Illustration of building blocks in the channel-wise perspective](image)

3.1 Rethink Shuffle Block in ShuffleNetV2

Essentially, residual connection is an element-wise operation with low computational consumption but high memory access cost. Distinct from common ConvNets, ShuffleNetV2\[23\] abandons residual connection to introduce short paths for neural network. Instead, to pursue higher efficiency, it combines "Channel Split" and "Channel Shuffle" to achieve cross-layer information communication, generating a proportion of short paths from the perspective of backpropagation.

Illustrated in Figure 1a, ShuffleNetV2 adopts split-transform-fuse strategy to design basic building block named shuffle block. The whole pipeline of shuffle block can be concluded as three stages:
**Split:** Channels of feature maps are split into two parts. One part is conserved in memory, another is fed into the next stage for high-level information extraction. Note that the conserved part is named "Feature Bank" in this paper. The number of feature maps in "Feature Bank" is determined by the parameter split ratio. It is seen that split ratio is a fixed parameter 0.5 in ShuffleNetV2.

**Transform:** Transformation stage consists of three layers: $1 \times 1$ regular convolution, $3 \times 3$ depth-wise convolution and regular $1 \times 1$ convolution in order, serving as a learnable "Feature Extractor".

**Fuse:** In this stage, feature maps coming from "Feature Bank" and previous stage are merged by concatenating along channel-wise dimension. Then, those merged channels are rearranged in an interleaving way, which is called "Channel Shuffle". "Channel Shuffle" enables information communication between the two branches.

Essentially, "Feature Bank" can be regarded as a auxiliary memory component where feature maps are stored temporarily. The success of ShuffleNetV2 lies in its use of "Feature Bank" via which to control the information communication across different layers. Here, "Channel Split" and "Channel Shuffle" are two main interactive operations between "Feature Bank" and "Feature Extractor". "Channel Split" determines the proportion of input feature maps which should be stored. "Channel Shuffle" is used to fuse old and new feature maps for the next building block. As a result, partial channels from shallow layers can be stored in "Feature Bank" and periodically fused into deeper layers of the network, introducing more short paths to replace residual connection [4] and achieving a similar effect of feature reuse [22].

Benefited from its unique design pattern, ShuffleNetV2 shows expressive balance between performance and efficiency. However, we notice that there still exists remaining room to improve for ShuffleNetV2:

1. It is seen that information communication is only limited to the exit of the shuffle block via the operation of "Channel Shuffle". Intermediate information generated by the first and second layer of shuffle block can not be fully exploited by other building blocks.

2. In the shuffle block of ShuffleNetV2, three convolutional layers are forced to keep the same number of input and output channels. As illustrated in Figure 2, they follows fully connected topological pattern in the channel-wise dimension. When network width (the number of channels) increases drastically, shuffle block in ShuffleNetV2 will produce more channel-wise redundancy, which constrains the design space of ShuffleNetV2 for better efficiency.

![Figure 2](image_url)

### 3.2 Augmented Shuffle Block

We propose another more powerful and efficient shuffle block illustrated in Figure 1b. Compared to the original shuffle block shown in 1a this paper makes following modifications:
M1. Split ratio $r$ is set as a variable, which can flexibly adjust the number of channels fed into "Feature Extractor" and further control the efficiency of the whole building block in terms of computational cost, parameter consumption and inference speed. For simplicity, the first and second layers ($1 \times 1$ regular convolution and $3 \times 3$ depth-wise convolution) keep the same number of input and output channels.

M2. We introduce another operation termed "Channel Crossover" to exchange information after the depth-wise convolutional layer. "Channel Crossover" deposit partial new feature maps into "Feature Bank" and withdraw more old feature maps to achieve channel-wise expansion without computational cost. By adding interaction mechanism between intermediate layers and "Feature Bank", information is fully utilized.

M3. We remove ReLU function after the first convolutional layer. M1, M2 and M3 work together to improve efficiency and make model gain better representational ability. when $r < 0.5$, the first and second layers in shuffle block would be more efficient than original shuffle block

3.3 Efficiency Analysis

Suppose both of shuffle blocks take $M$ feature maps with a spatial size of $D_f \times D_f$ as input. The split ratio is set to $r$, the kernel size of depth-wise convolution is defined as $D_k$. Hence, the computational cost of augmented shuffle block can be formulated as

$$C = r^2 M^2 D_f^2 + r M D_f^2 D_k^2 + \frac{1}{4} M^2 D_f^2$$  \hspace{1cm} (1)

The computational cost of original shuffle block in ShuffleNetV2 can be also obtained via Equation (1) we only need to set $r$ to 0.5. Therefore, the ratio of computational cost of two shuffle blocks in AugShuffleNet and ShuffleNetV2 respectively is

$$\alpha = \frac{C_r}{C_{r=0.5}} = \frac{(4r^2 + 1) M + 4r D_k^2}{2M + 2D_k^2} \approx \frac{4r^2 + 1}{2}$$  \hspace{1cm} (2)

As depth-wise convolution is a quite efficient operator, its overhead can be ignored. Here, we can obtain an approximate expression of $\alpha$. It is seen $\alpha$ is almost only determined by split ratio $r$. It means that split ratio $r$ can be used as a global hyper-parameter to reduce the high-dimensional channel-wise redundancy from two $1 \times 1$ convolutional layers and adjust the efficiency of AugShuffleNet.

Similarly, the parameter consumption $P$ of augmented shuffle block in AugShuffleNet can be formulated as

$$P = r^2 M^2 + r M D_k^2 + \frac{1}{4} M^2$$  \hspace{1cm} (3)

the ratio of parameter consumption is also equal to $\alpha$:

$$\beta = \frac{P_r}{P_{r=0.5}} = \frac{(4r^2 + 1) M + 4r D_k^2}{2M + 2D_k^2} \approx \frac{4r^2 + 1}{2}$$  \hspace{1cm} (4)

In general, as $r$ decreases, augmented shuffle block becomes more efficient compared to original shuffle block in ShuffleNetV2.

4 Network Design

The architecture of AugShuffleNet is shown in Table (1). We offer three models with different network width ($0.5 \times$, $1.0 \times$ and $1.5 \times$). To ensure fair comparison, we follow the architecture configuration of ShuffleNetV2. The only difference is that we set the number of channels for Stage2, Stage3 and Stage 4 in AugShuffleNet $1.0 \times$ to 120, 240 and 480 while ShuffleNetV2 $1.0 \times$‘s are 116, 232 and 464. As we need proper channel partition to explore the effect of different split ratio. Every stage in Table (1) includes a down-sampling block (Stride = 2) and repeated normal blocks (Stride = 1). In this paper, we adopt original down-sampling block in ShuffleNetV2 and use augmented shuffle block illustrated in Figure (1b) as normal block to build AugShuffleNet. The split ratio $r$ of all of augmented shuffle blocks are forced to keep the same for AugShuffleNet. The default value of $r$ is set to 0.375.
Table 1: Architecture of AugShuffleNet

| Layer     | Output size | KSize | Stride | Repeat | Output channels |
|-----------|-------------|-------|--------|--------|-----------------|
| Image     | 32 × 32     |       |        |        | 0.5× 1× 1.5×    |
| Conv      | 32 × 32     | 3 × 3 | 1      | 1      | 24              |
| Stage2    | 16 × 16     | 3 × 3 | 1      | 1      | 48              |
| Stage3    | 8 × 8       | 3 × 3 | 1      | 1      | 96              |
| Stage4    | 4 × 4       | 3 × 3 | 1      | 1      | 192             |
| Conv      | 4 × 4       | 1 × 1 | 1      | 1      | 1024            |
| GlobalPool| 1 × 1       | 4 × 4 |        |        | num_classes     |
| FC        |             |       |        |        | num_classes     |

5 Experiment and Result

Datasets All models are evaluated on the datasets CIFAR-10 and CIFAR-100. The two CIFAR datasets consist of colored natural images with 32 × 32 pixels. CIFAR-10 and CIFAR-100 consist of images drawn from 10 classes and 100 classes respectively. The training and test sets contain 50,000 and 10,000 images respectively for both two datasets. Images in the training set are augmented by random horizontal flip and random crop (4 pixels are padded on each side, and a 32×32 crop is randomly sampled from the padded image).

Training Settings All models are trained 300 epochs using cosine learning rate decay for both of CIFAR-10 and CIFAR-100. The initial learning rate is set as 0.1. we adopt Stochastic Gradient Descend (SGD) optimizer (momentum parameter is 0.9, nesterov is set to True) with the batch size of 128. The weight decay is set to 1e-4.

Table 2: Comparison of models on CIFAR-10

| Model         | Params | FLOPs | #run1 | #run2 | #run3 | #run4 | #run5 | #Average |
|---------------|--------|-------|-------|-------|-------|-------|-------|----------|
| ShuffleNetV2 1.5×| 2.49M  | 94.27M| 93.93 | 93.96 | 93.86 | 93.98 | 94.11 | 93.97    |
| AugShuffleNet 1.5×| 2.22M  | 85.38M| 94.31 | 94.41 | 94.21 | 94.44 | 94.62 | 94.40    |
| ShuffleNetV2 1.0×| 1.26M  | 45.01M| 93.12 | 93.21 | 93.17 | 93.30 | 93.42 | 93.24    |
| AugShuffleNet 1.0×| 1.21M  | 43.56M| 93.63 | 93.78 | 93.95 | 93.96 | 93.87 | 93.84    |
| ShuffleNetV2 0.5×| 0.35M  | 10.91M| 90.62 | 90.95 | 90.67 | 90.79 | 91.12 | 90.83    |
| AugShuffleNet 0.5×| 0.33M  | 10.20M| 90.85 | 90.87 | 91.21 | 91.39 | 90.92 | 91.05    |

Table 3: Comparison of models on CIFAR-100

| Model         | Params | FLOPs | #run1 | #run2 | #run3 | #run4 | #run5 | #Average |
|---------------|--------|-------|-------|-------|-------|-------|-------|----------|
| ShuffleNetV2 1.5×| 2.58M  | 94.36M| 74.53 | 73.46 | 74.83 | 74.76 | 74.70 | 74.70    |
| AugShuffleNet 1.5×| 2.32M  | 85.47M| 75.65 | 75.75 | 75.98 | 76.44 | 75.75 | 75.91    |
| ShuffleNetV2 1.0×| 1.36M  | 45.10M| 73.24 | 73.06 | 72.91 | 72.70 | 73.49 | 73.08    |
| AugShuffleNet 1.0×| 1.30M  | 43.65M| 74.07 | 74.10 | 74.31 | 74.45 | 73.88 | 74.16    |
| ShuffleNetV2 0.5×| 0.44M  | 11.00M| 68.97 | 68.46 | 68.97 | 68.39 | 68.67 | 68.69    |
| AugShuffleNet 0.5×| 0.42M  | 10.29M| 69.19 | 69.03 | 69.29 | 69.93 | 69.35 | 69.15    |

5.1 Comparison to Other Models

To verify our approach, we choose ShuffleNetV2 as baselines. Accuracy, computational cost and parameter count are selected as evaluation metrics. Note that, term FLOPs means multiply-adds (MAdd) in this paper. Extensive experiments are conducted on image classification datasets CIFAR-10 and CIFAR-100. The result for every model are obtained via 5 runs to ensure the reliability of experiments. Result are reported in Table 2 and Table 3. It is clear that our models consistently outperform ShuffleNetV2 model in accuracy aspect on both CIFAR-10 and CIFAR-100 with
better efficiency in terms of computational cost and parameter count. The results under different network width (0.5×, 1.0× and 1.5×) is sufficient to demonstrate the advantage of augmented shuffle block in AugShuffleNets.

5.2 Ablation Study

Split ratio $r$ is an important hyper-parameter for AugShuffleNet, which directly determines the efficiency of augmented shuffle block. To explore the impact of split ratio $r$, we compare the performance and efficiency between AugShuffleNets 1.5× with different split ratio (0.125, 0.25 and 0.375) and ShuffleNetV2 1.5×.

As illustrated in Figure 3 and Figure 4, all AugShuffleNet 1.5× variants obtain higher accuracy than ShuffleNetV2 1.5× with less computational cost and fewer parameters on both CIFAR-10 and CIFAR-100. It is noteworthy that AugShuffleNet 1.5× ($r = 0.125$) is 0.29% (0.91%) more accurate than AugShuffleNet 1.5× with 79.6% (79.6%) computational cost and 76.7% (77.9%) parameter count on CIFAR-10 (CIFAR-100).

The above results indicate that the channel-wise redundancy in AugShuffleNet 1.5× is quite considerable for small datasets CIFAR-10 and CIFAR-100. This kind of redundancy problem results from the design pattern of original shuffle block. Our augmented shuffle block is more flexible in adjusting the overhead of the whole building block, finally reducing the channel-wise redundancy due to large network width.

Figure 3: Comparison between AugShuffleNets 1.5× with different split ratio (0.125, 0.25 and 0.375) and ShuffleNetV2 1.5× on CIFAR-10.
Figure 4: Comparison between AugShuffleNets 1.5× with different split ratio (0.125, 0.25 and 0.375) and ShuffleNetV2 1.5× on CIFAR-100.

6 Conclusion

We propose a more flexible and efficient building block named augmented shuffle block based on ShuffleNetV2. By replacing original shuffle block in ShuffleNetV2, we can see obvious improvement in both performance and efficiency. Our experimental results show that augmented shuffle block is a potential building block to construct compact models for resource-constrained platforms.

References

[1] 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.
[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
[3] 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
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.

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

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

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

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.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

Andrey Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

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

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

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.

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

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

Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4013–4021, 2016.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, 29:550–558, 2016.

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

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.