MobileOne: An Improved One millisecond Mobile Backbone

Tuan-Anh Bui[1]

[1]Department of Data Science and AI
Faculty of Information Technology
Monash University

Paper Reading

Table of Contents

Table of Contents

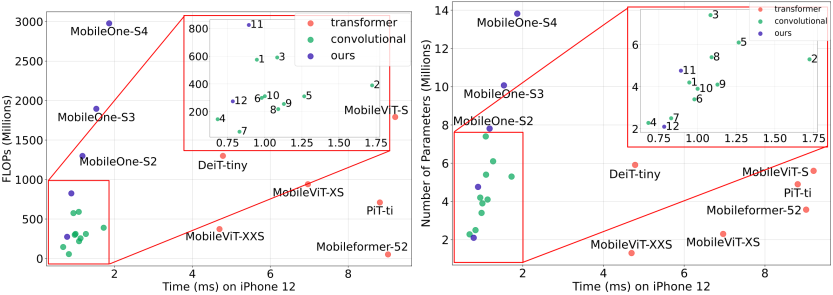
Motivation

Goal: To improve the efficiency of DNNs on mobile devices
Most prior works [1], [2] focus on improving the efficiency of DNNs by reducing the number of FLOPS and parameters.
However, the relationship between latency, #FLOPS, and #Parameters is not well understood, especially when deploying on real devices.
Contributions:

- First, empirically study the key bottlenecks of deploying DNNs on mobile devices. Found that activation functions and model architecture are two key factors.
- Second, propose a new model, MobileOne, that improves the latency of DNNs on mobile devices.

Motivation

The empirical study shows that there is no linear relationship between #FLOPS/#Params and latency.



1 MobileNetV1	2 EfficientNet-B0	3 ShuffleNetV2-2.0	4 ShuffleNetV2-1.0
5 MobileNext-1.4	6 MobileNetV2	7 MobileNetV3-S	8 MobileNetV3-L
9 MixNet-S	10 MNASNet-A1	11 MobileOne-S1	12 MobileOne-S0

Figure: Left: FLOPs vs Latency on iPhone12. Right: Parameter Count vs Latency on iPhone 12. Some networks are indicating by numbers as shown in the table above.

Key Bottlenecks

Comparison of latency on mobile device of different activation functions in a 30-layer convolutional neural network.

Activation Function	Latency (ms)
ReLU [3]	1.53
GELU [4]	1.63
SE-ReLU [5]	2.10
SiLU [6]	2.54
Dynamic Shift-Max [7]	57.04
DynamicReLU-A [8]	273.49
DynamicReLU-B [8]	242.14

Ablation on latency of different architectural blocks in a 30-layer convolutional neural network.

Architectural Blocks	Baseline	+ Squeeze Excite [5]	+ Skip Connections [9]
Latency (ms)	1.53	2.10	2.62

Multiple branches (e.g., ResNet) requires more memory to buffer intermediate features, leading to higher latency [2].

Table of Contents

Structural Re-parameterization

The key idea is to use multiple branches at train-time and re-parameterize them into a single branch at inference time [1], [2].

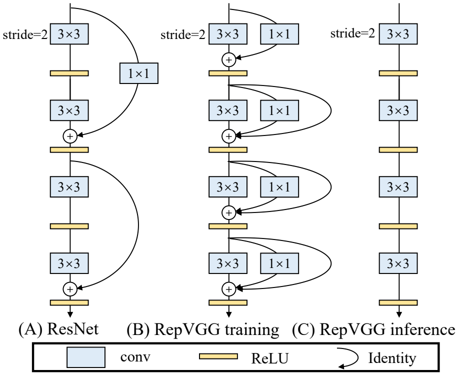


Figure: Sketch of RepVGG architecture. RepVGG has 5 stages and conducts down-sampling via stride-2 convolution at the beginning of a stage

Structural Re-parameterization

The training time block is $y = x + g(x) + f(x)$ if the output has the same shape as the input x, and $y = g(x) + f(x)$ otherwise, where g and f are two branches.

$$\begin{aligned}\mathrm{M}^{(2)} &= \mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(3)}, \mu^{(3)}, \sigma^{(3)}, \gamma^{(3)}, \beta^{(3)}) \\ &+ \mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(1)}, \mu^{(1)}, \sigma^{(1)}, \gamma^{(1)}, \beta^{(1)}) \\ &+ \mathrm{bn}(\mathrm{M}^{(1)}, \mu^{(0)}, \sigma^{(0)}, \gamma^{(0)}, \beta^{(0)}) \,. \end{aligned} \tag{1}$$

and bn is the inference-time BN function:

$$\mathrm{bn}(\mathrm{M}, \mu, \sigma, \gamma, \beta)_{:,i,:,:} = (\mathrm{M}_{:,i,:,:} - \mu_i)\frac{\gamma_i}{\sigma_i} + \beta_i \,. \tag{2}$$

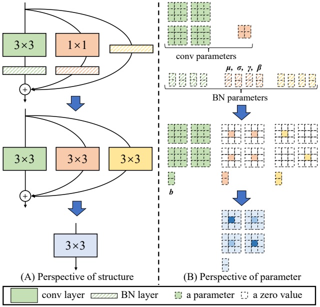


Figure: Structural re-parameterization of a RepVGG block.

Where $\mathrm{M}^{(1)} \in \mathbb{R}^{N \times C_1 \times H_1 \times W_1}$, $\mathrm{M}^{(2)} \in \mathbb{R}^{N \times C_2 \times H_2 \times W_2}$ be the input and output, respectively, and μ, σ, γ, β are the mean, variance, scale, and shift parameters of the BN layer.

Structural Re-parameterization

The key idea is to use multiple branches at train-time and re-parameterize them into a single branch at inference time.[1]

BN layer can be folded into the preceding convolutional layer.

$$\mathrm{bn}(\mathrm{M}*\mathrm{W}, \mu, \sigma, \gamma, \beta)_{:,i,:,:} = (\mathrm{M}*\mathrm{W}')_{:,i,:,:} + \mathbf{b}'_i \,. \quad (3)$$

where $\{\mathrm{W}', \mathbf{b}'\}$ are the re-parameterized weights and biases:

$$\mathrm{W}'_{:,i,:,:} = \frac{\gamma_i}{\sigma_i}\mathrm{W}_{:,i,:,:} \,, \quad \mathbf{b}'_i = -\frac{\mu_i \gamma_i}{\sigma_i} + \beta_i \,. \quad (4)$$

After folding, we have one 3×3 kernel, two 1×1 kernels, and one bias term. We then can merge three kernels into a single 3×3 one.

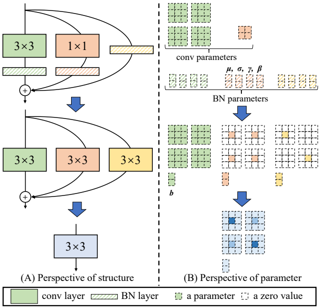


Figure: Structural re-parameterization of a RepVGG block.

[1] This requires output of 3 branches to have the same shape as the input.

Table of Contents

MobileOne Block

The MobileOne block has two different structures at train time and test time. k is the over-parameterization factor (repeat factor) which is a hyper-parameter to be tuned.
And the Conv and BN can be merged into a single convolutional layer as in RepVGG

$$\begin{aligned} W' &= W * \gamma/\sigma, \\ b' &= -\mu * \gamma/\sigma + \beta \end{aligned} \qquad (5)$$

$$b' = (b - \mu) * \gamma/\sigma + \beta \text{ ,if } (b \neq 0)$$

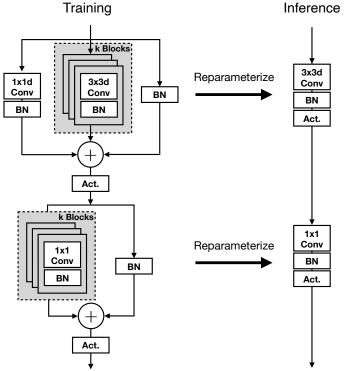


Figure: MobileOne block has two different structures at train time and test time.

Tuning k: it helps on small models but does not affect much on large models.

Model	Top-1				
	k=1	k=2	k=3	k=4	k-5
MobileOne-S0	70.9	70.7	71.3	71.4	71.1
MobileOne-S1	75.9	75.7	75.6	75.6	75.2

MobileOne Network Specifications, where α is the depth multiplier.

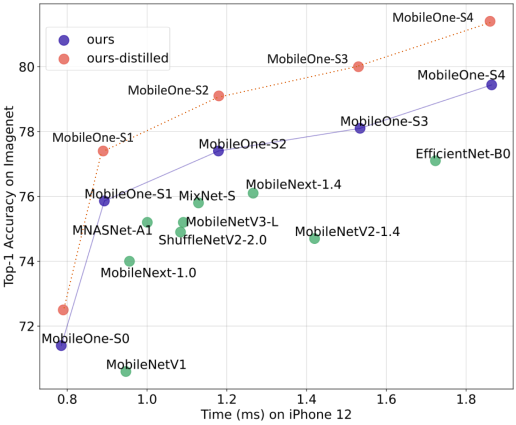
Stage	Input	# Blocks	Stride	Block Type	# Channels	MobileOne Block Parameters (α, k, act=ReLU)				
						S0	S1	S2	S3	S4
1	224×224	1	2	MobileOne-Block	$64\times\alpha$	(0.75, 4)	(1.5, 1)	(1.5, 1)	(2.0, 1)	(3.0, 1)
2	112×112	2	2	MobileOne-Block	$64\times\alpha$	(0.75, 4)	(1.5, 1)	(1.5, 1)	(2.0, 1)	(3.0, 1)
3	56×56	8	2	MobileOne-Block	$128\times\alpha$	(1.0, 4)	(1.5, 1)	(2.0, 1)	(2.5, 1)	(3.5, 1)
4	28×28	5	2	MobileOne-Block	$256\times\alpha$	(1.0, 4)	(2.0, 1)	(2.5, 1)	(3.0, 1)	(3.5, 1)
5	14×14	5	1	MobileOne-Block	$256\times\alpha$	(1.0, 4)	(2.0, 1)	(2.5, 1)	(3.0, 1)	(3.5, 1, SE-ReLU)
6	14×14	1	2	MobileOne-Block	$512\times\alpha$	(2.0, 4)	(2.5, 1)	(4.0, 1)	(4.0, 1)	(4.0, 1, SE-ReLU)
7	7×7	1	1	AvgPool	-	-	-	-	-	-
8	1×1	1	1	Linear	$512\times\alpha$	2.0	2.5	4.0	4.0	4.0

MobileOne Block - Experimental Results

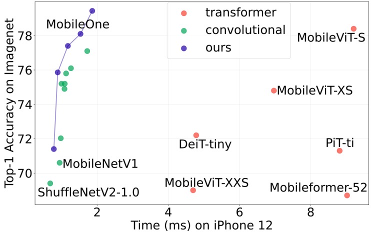
Model	# Params.	Top-1
ExpandNet-CL MobileNetV1 [10]	4.2	69.4
RepVGG-A0 [11]	8.3	72.4
RepVGG-A1 [11]	12.8	74.5
RepVGG-B0 [11]	14.3	75.1
ACNet MobileNetV1 [12]	4.2	72.1
ACNet ResNet18 [12]	11.7	71.1
DBBNet MobileNetV1 [13]	4.2	72.9
DBBNet ResNet18 [13]	11.7	71.0
MobileOne-S0	2.1	71.4
MobileOne-S1	4.8	75.9
MobileOne-S2	7.8	77.4
MobileOne-S3	10.1	78.1
MobileOne-S4	14.8	79.4

Table: Comparison of Top-1 Accuracy on ImageNet against recent train time over-parameterization works. Number of parameters listed above is at inference.

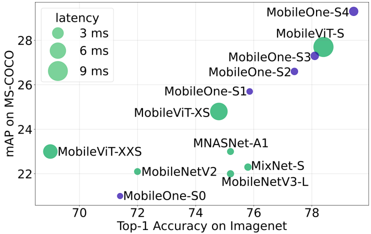
MobileOne Block - Experimental Results



(a) Top 1 accuracy vs Latency on iPhone 12.



(b) Zoomed out (a)



(c) Top-1 accuracy vs mAP.

MobileOne Block - Experimental Results

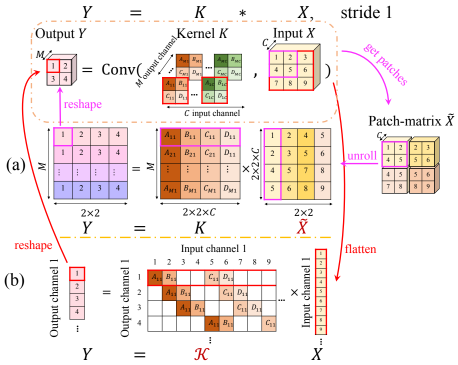
Benchmarking latency on real devices (iPhone 12). Note that, the authors could not breakdown the latency into sub-components, i.e., network initialization, data loading, model execution, etc. Therefore, a large fraction of the time may be from platform processes but not the model execution. The authors also benchmarked on CPU (Ubuntu desktop with Intel processor) and GPU (RTX-2080Ti, batch size 1 using TensorRT).

- Table 9. Transformer models quite slow on Mobile, even with smaller # Params.
- Table 9. MobileNeXt, MobileNetV3-Large, and ShuffleNetsV2 are quite good baselines but not the best as MobileOne.
- Table 9. Other methods also achieve millisecond latency so the title is a bit overselling.
- Table 11. MobileOne is also good at object detection and segmentation tasks.

Table of Contents

Convolution as a Matrix Multiplication

A convolutional layer $Y = \text{Conv}(K, X)$ can be formulated as matrix multiplications in two ways: **a)** *im2col* methods [14], [15] retain kernel K and convert input X to patch-matrix $\widetilde{X}$. **b)** Retaining input X and convert K to a doubly block-Toeplitz matrix $\mathcal{K}$. With X and Y intact, we directly analyze the transformation from the input to the output [16].



Convolution as a Matrix Multiplication

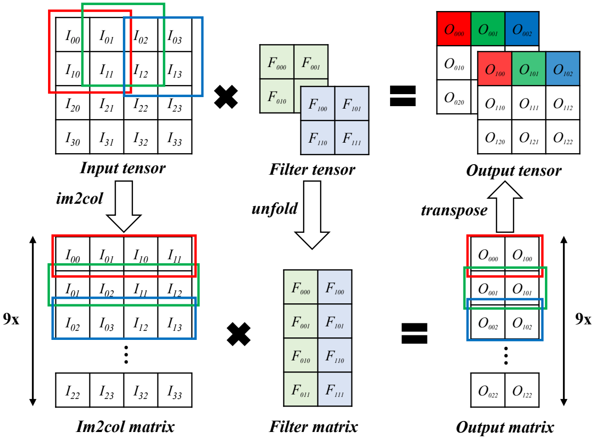
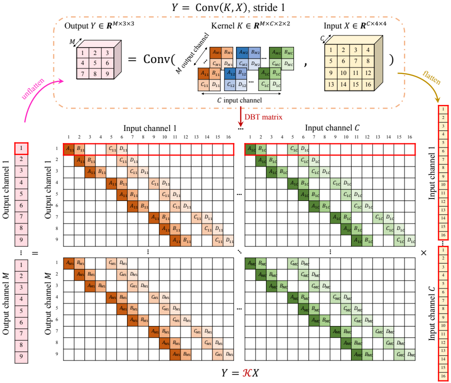


Figure: Basic convolution and im2col+GEMM convolution examples with $H_i = W_i = 4$, $H_f = W_f = 2$, $s_h = s_w = 1$, $H_o = W_o = 3$ ($C_i = C_f = 1$, $C_o = 2$). The different colored boxes denote the correspondence between the input image and the output features.

Convolution as a Matrix Multiplication

Convolution based on the doubly block-Toeplitz (DBT) matrix. We first flatten X to a vector $\mathbf{x}$, and then convert weight tensor $K \in \mathbf{R}^{M\times C\times k\times k}$ as DBT matrix $\mathcal{K} \in \mathbf{R}^{(MH'W')\times(CHW)}$. The output $\mathbf{y} = \mathcal{K}\mathbf{x}$. We can obtain the desired output $Y \in \mathbf{R}^{M\times H'\times W'}$ by reshaping $\mathbf{y}$. The example has input size $C \times 4 \times 4$, kernel size $M \times C \times 2 \times 2$ and stride 1



Convolution as a Matrix Multiplication - Low-rank Approximation

Folding, converting to matrix multiplication, and low-rank approximation can be combined together to reduce the number of parameters and FLOPs for the convolutional layer.

Prior work [17] has shown that low-rank approximation reduce the number of parameters and FLOPs for DNNs.

So, combining Folding, converting to matrix multiplication, and low-rank approximation can be a promising direction.

Table 3. Compression statistics for VGG-16. L: Low-rank. S: Sparse. R: Compression rate.

Layer	#W	#L/#W	#S/#W	R
conv1_1	2K	0%	100%	100%
conv1_2	37K	10%	10%	20%
conv2_1	74K	12%	11%	23%
conv2_2	148K	11%	10%	23%
conv3_1	295K	12%	12%	24%
conv3_2	590K	11%	11%	22%
conv3_3	590K	11%	11%	22%
conv4_1	1M	12%	12%	24%
conv4_2	2M	11%	11%	22%
conv4_3	2M	11%	11%	22%
conv5_1	2M	11%	11%	22%
conv5_2	2M	11%	11%	22%
conv5_3	2M	11%	11%	22%

Diversifying Branches

Encouraging diversity/orthogonalization among branches can improve the model performance, e.g., [16].

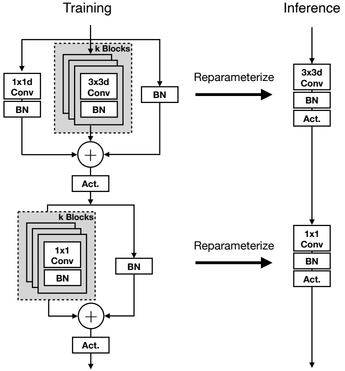


Figure: MobileOne block has two different structures at train time and test time.

Im2Win Data Transformation

Im2Col transforms convolution into matrix multiplication by converting the input tensor to a matrix, followed by a GEMM function call using Basic Linear Algebra Subprograms (BLAS) [18]. However, Im2Col has a high memory footprint due to the large size of the matrix.

Direct convolution has no additional memory overhead but its memory access is nonconsecutive, leading to poor cache utilization and data reuse. The larger the kernel size, the worse the cache utilization.

Im2Win is a new data transformation method which enables data reuse and cache utilization.

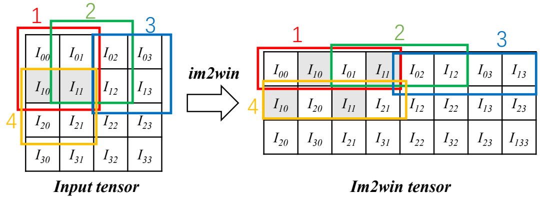


Figure: Illustration of the im2win data transformation from the original tensor to the im2win tensor. The different colored boxes indicate the mapping of elements between the input and the im2win tensors. After transformation, the im2win tensor has 24 elements.

Im2Win Data Transformation

Im2Win is a new data transformation method which enables data reuse and cache utilization.

This method reduces the memory overhead by average to 41.6% compared to the PyTorch's up-to-date convolution implementation based on im2col, and achieves on average 3.6× and 5.3× speedup in performance compared to the im2col-based convolution and not using data transformation, respectively.

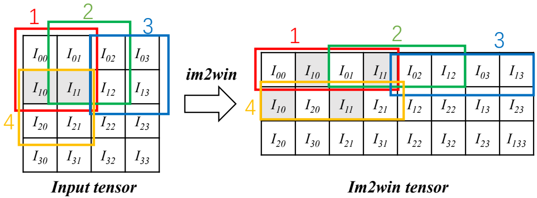


Figure: Illustration of the im2win data transformation from the original tensor to the im2win tensor. The different colored boxes indicate the mapping of elements between the input and the im2win tensors. After transformation, the im2win tensor has 24 elements.

[1] S. Zagoruyko and N. Komodakis, "Diracnets: Training very deep neural networks without skip-connections," *arXiv preprint arXiv:1706.00388*, 2017.

[2] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "Repvgg: Making vgg-style convnets great again," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 13 733–13 742.

[3] A. F. Agarap, "Deep learning using rectified linear units (relu)," *Neural and Evolutionary Computing*, 2018.

[4] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[5] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[6] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural Networks*, vol. 107, pp. 3–11, 2018.

[7] Y. Li, Y. Chen, X. Dai, *et al.*, "Micronet: Improving image recognition with extremely low flops," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[8] Y. Chen, X. Dai, M. Liu, D. Chen, L. Yuan, and Z. Liu, "Dynamic relu," in *16th European Conference Computer Vision (ECCV 2020)*, 2020.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[10] S. Guo, J. M. Alvarez, and M. Salzmann, "Expandnets: Linear over-parameterization to train compact convolutional networks," in *Advances in Neural Information Processing Systems*, 2020.

[11] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "Repvgg: Making vgg-style convnets great again," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[12] X. Ding, Y. Guo, G. Ding, and J. Han, "Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2019.

[13] X. Ding, X. Zhang, J. Han, and G. Ding, "Diverse branch block: Building a convolution as an inception-like unit," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[14] K. Yanai, R. Tanno, and K. Okamoto, "Efficient mobile implementation of a cnn-based object recognition system," in *Proceedings of the International Conference on Multimedia (ICM)*, 2016, pp. 362–366.

[15] F. Heide, W. Heidrich, and G. Wetzstein, "Fast and flexible convolutional sparse coding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 5135–5143.

[16] J. Wang, Y. Chen, R. Chakraborty, and S. X. Yu, "Orthogonal convolutional neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 505–11 515.

[17] X. Yu, T. Liu, X. Wang, and D. Tao, "On compressing deep models by low rank and sparse decomposition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7370–7379.

[18] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff, "A set of level 3 basic linear algebra subprograms," *ACM Transactions on Mathematical Software (TOMS)*, vol. 16, no. 1, pp. 1–17, 1990.