Run, Don't Walk: Chasing Higher FLOPS for Faster Neural Networks – Supplementary Material –

Jierun Chen[1], Shiu-hong Kao[1], Hao He[1]
Weipeng Zhuo[1], Song Wen[2], Chul-Ho Lee[3], S.-H. Gary Chan[1]
[1]HKUST, [2]Rutgers University, [3]Texas State University

In this appendix, we provide further details on the experimental settings, full comparison plots, architectural configurations, PConv implementations, comparisons with related work, limitations, and future work.

A. ImageNet-1k experimental settings

We provide ImageNet-1k training and evaluation settings in Tab. 6. They can be used for reproducing our main results in Tab.3 and Fig.7. Different FasterNet variants vary in the magnitude of regularization and augmentation techniques. The magnitude increases as the model becomes larger to alleviate overfitting and improve accuracy. Note that most of the compared works in Tab.3 and Fig.7, *e.g.*, MobileViT, EdgeNext, PVT, CycleMLP, ConvNeXt, Swin, etc., also adopt such advanced training techniques (ADT). Some even heavily rely on the hyper-parameter search. For others w/o ADT, *i.e.*, ShuffleNetV2, MobileNetV2, and GhostNet, though the comparison is not totally fair, we include them for reference.

B. Downstream tasks experimental settings

For object detection and instance segmentation on the COCO2017 dataset, we equip our FasterNet backbone with the popular Mask R-CNN detector. We use ImageNet-1k pre-trained weights to initialize the backbone and Xavier to initialize the add-on layers. Detailed settings are summarized in Tab. 7.

C. Full comparison plots on ImageNet-1k

Fig. 8 shows the full comparison plots on ImageNet-1k, which is the extension of Fig.7 in the main paper with a larger range of latency. Fig. 8 shows consistent results that FasterNet strikes better trade-offs than others in balancing accuracy and latency/throughput on GPU, CPU, and ARM processors.

D. Detailed architectural configurations

We present the detailed architectural configurations in Tab. 8. While different FasterNet variants share a unified architecture, they vary in the network width (the number of channels) and network depth (the number of FasterNet blocks at each stage). The classifier at the end of the architecture is used for classification tasks but removed for other downstream tasks.

E. More comparisons with related work

Improving FLOPS. There are a few other works [1, 11] also looking into the FLOPS issue and trying to improve it. They generally follow existing operators and try to find their proper configurations, *e.g.*, RepLKNet [1] simply increases the kernel size while TRT-ViT [11] reorders different blocks in the architecture. By contrast, this paper advances the field by proposing a novel and efficient PConv, opening up new directions and potentially larger room for FLOPS improvement.

PConv vs. GConv. PConv is schematically equivalent to a modified GConv [6] that operates on a single group and leaves other groups untouched. Though simple, such a modification remains unexplored before. It's also significant in the sense that it prevents the operator from excessive memory access and is computationally more efficient. From the perspective of low-rank approximations, PConv improves GConv by further reducing the intra-filter redundancy beyond the inter-filter redundancy [4].

FasterNet vs. ConvNeXt. Our FasterNet appears similar to ConvNeXt [9] after substituting DWConv with our PConv. However, they are different in motivations. While ConvNeXt searches for a better structure by trial and error, we append PWConv after PConv to better aggregate information from all channels. Moreover, ConvNeXt follows ViT to use fewer activation functions, while we intentionally remove them from the middle of PConv and PWConv, to minimize their error in approximating a regular Conv.

Variants	T0	T1	T2	S	M	L
Train Res	192 for epoch 1∼280, 224 for epoch 281∼300					
Test Res	224					
Epochs	300					
# of forward pass	188k					
Batch size	4096	4096	4096	4096	2048	2048
Optimizer	AdamW					
Momentum	0.9/0.999					
LR	0.004	0.004	0.004	0.004	0.002	0.002
LR decay	cosine					
Weight decay	0.005	0.01	0.02	0.03	0.05	0.05
Warmup epochs	20					
Warmup schedule	linear					
Label smoothing	0.1					
Dropout	✗					
Stoch. Depth	✗	0.02	0.05	0.1	0.2	0.3
Repeated Aug	✗					
Gradient Clip.	✗	✗	✗	✗	1	0.01
H. flip	✓					
RRC	✓					
Rand Augment	✗	3/0.5	5/0.5	7/0.5	7/0.5	7/0.5
Auto Augment	✗					
Mixup alpha	0.05	0.1	0.1	0.3	0.5	0.7
Cutmix alpha	1.0					
Erasing prob.	✗					
Color Jitter	✗					
PCA lighting	✗					
SWA	✗					
EMA	✗					
Layer scale	✗					
CE loss	✓					
BCE loss	✗					
Mixed precision	✓					
Test crop ratio	0.9					
Top-1 acc. (%)	71.9	76.2	78.9	81.3	83.0	83.5

Table 6. ImageNet-1k training and evaluation settings for different FasterNet variants.

Variants	S	M	L
Train and test Res	shorter side $=$ 800, longer side $\leq$ 1333		
Batch size	16 (2 on each GPU)		
Optimizer	AdamW		
Train schedule	1$\times$ schedule (12 epochs)		
Weight decay	0.0001		
Warmup schedule	linear		
Warmup iterations	500		
LR decay	StepLR at epoch 8 and 11 with decay rate 0.1		
LR	0.0002	0.0001	0.0001
Stoch. Depth	0.15	0.2	0.3

Table 7. Experimental settings of object detection and instance segmentation on the COCO2017 dataset.

Other paradigms for efficient inference. Our work focuses on efficient network design, orthogonal to the other paradigms, *e.g.*, neural architecture search (NAS) [2], network pruning [10], and knowledge distillation [5]. They can be applied in this paper for better performance. However, we opt not to do so to keep our core idea centered and to make the performance gain clear and fair.

Other partial/masked convolution works. There are several works [3, 7, 8] sharing similar names with our PConv. However, they differ a lot in objectives and methods. For example, they apply filters on partial pixels to exclude invalid patches [8], enable self-supervised learning [3], or synthesize novel images [7], while we target at the channel dimension for efficient inference.

F. Limitations and future work

We have demonstrated that PConv and FasterNet are fast and effective, being competitive with existing operators and networks. Yet there are some minor technical limitations of this paper. For one thing, PConv is designed to apply a regular convolution on only a part of the input channels while leaving the remaining ones untouched. Thus, the stride of the partial convolution should always be 1, in order to align the spatial resolution of the convolutional output and that of the untouched channels. Note that it is still feasible to down-sample the spatial resolution as there can be additional downsampling layers in the architecture. And for another, our FasterNet is simply built upon convolutional operators with a possibly limited receptive field. Future efforts can be made to enlarge its receptive field and combine it with other operators to pursue higher accuracy.

References

[1] Xiaohan Ding et al. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *CVPR*, 2022. 1

[2] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019. 2

[3] Peng Gao, Teli Ma, Hongsheng Li, Jifeng Dai, and Yu Qiao. Convmae: Masked convolution meets masked autoencoders. *arXiv preprint arXiv:2205.03892*, 2022. 2

[4] Daniel Haase et al. Rethinking depthwise separable convolutions: How intra-kernel correlations lead to improved mobilenets. In *CVPR*, 2020. 1

[5] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015. 2

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. 1

[7] Guilin Liu, Aysegul Dundar, Kevin J Shih, Ting-Chun Wang, Fitsum A Reda, Karan Sapra, Zhiding Yu, Xiaodong Yang,

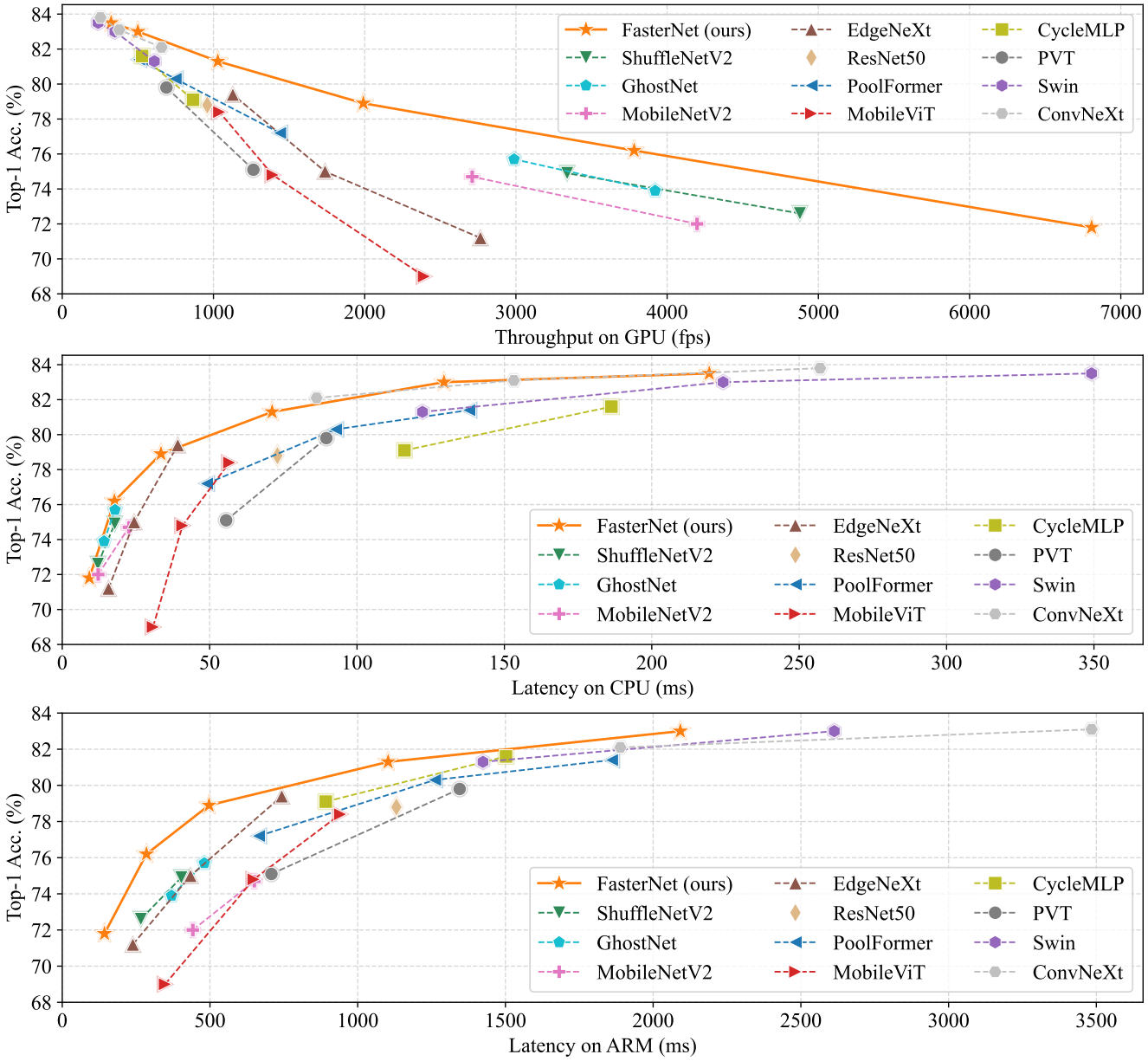

Figure 8. Comparison of FasterNet with state-of-the-art networks. FasterNet consistently achieves better accuracy-throughput (the top plot) and accuracy-latency (the medium and bottom plots) trade-offs than others.

Andrew Tao, and Bryan Catanzaro. Partial convolution for padding, inpainting, and image synthesis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. 2

[8] Guilin Liu, Fitsum A Reda, Kevin J Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. In *Proceedings of the European conference on computer vision (ECCV)*, pages 85–100, 2018. 2

[9] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022. 1

[10] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016. 2

[11] Xin Xia et al. Trt-vit: Tensorrt-oriented vision transformer. *arXiv preprint*, 2022. 1

Name	Output size	Layer specification		T0	T1	T2	S	M	L
Embedding	$\frac{h}{4} \times \frac{w}{4}$	Conv_4_c_4, BN	# Channels c	40	64	96	128	144	192
Stage 1	$\frac{h}{4} \times \frac{w}{4}$	$\begin{bmatrix} \text{PConv\_3\_}c\text{\_1\_1/4}, \\ \text{Conv\_1\_}2c\text{\_1}, \\ \text{BN, Acti,} \\ \text{Conv\_1\_}c\text{\_1} \end{bmatrix} \times b_1$	# Blocks b_1	1	1	1	1	3	3
Merging	$\frac{h}{8} \times \frac{w}{8}$	Conv_2_$2c$_2, BN	# Channels $2c$	80	128	192	256	288	384
Stage 2	$\frac{h}{8} \times \frac{w}{8}$	$\begin{bmatrix} \text{PConv\_3\_}2c\text{\_1\_1/4}, \\ \text{Conv\_1\_}4c\text{\_1}, \\ \text{BN, Acti,} \\ \text{Conv\_1\_}2c\text{\_1} \end{bmatrix} \times b_2$	# Blocks b_2	2	2	2	2	4	4
Merging	$\frac{h}{16} \times \frac{w}{16}$	Conv_2_$4c$_2, BN	# Channels $4c$	160	256	384	512	576	768
Stage 3	$\frac{h}{16} \times \frac{w}{16}$	$\begin{bmatrix} \text{PConv\_3\_}4c\text{\_1\_1/4}, \\ \text{Conv\_1\_}8c\text{\_1}, \\ \text{BN, Acti,} \\ \text{Conv\_1\_}4c\text{\_1} \end{bmatrix} \times b_3$	# Blocks b_3	8	8	8	13	18	18
Merging	$\frac{h}{32} \times \frac{w}{32}$	Conv_2_$8c$_2, BN	# Channels $8c$	320	512	768	1024	1152	1536
Stage 4	$\frac{h}{32} \times \frac{w}{32}$	$\begin{bmatrix} \text{PConv\_3\_}8c\text{\_1\_1/4}, \\ \text{Conv\_1\_}16c\text{\_1}, \\ \text{BN, Acti,} \\ \text{Conv\_1\_}8c\text{\_1} \end{bmatrix} \times b_4$	# Blocks b_4	2	2	2	2	3	3
Classifier	1×1	Global average pool, Conv_1_1280_1, Acti, FC_1000	Acti	GELU	GELU	ReLU	ReLU	ReLU	ReLU
FLOPs (G)				0.34	0.85	1.90	4.55	8.72	15.49
Params (M)				3.9	7.6	15.0	31.1	53.5	93.4

Table 8. Configurations of different FasterNet variants. "Conv_k_c_s" means a convolutional layer with the kernel size of k, the output channels of c, and the stride of s. "PConv_k_c_s_r" means a partial convolution with an extra parameter, the partial ratio of r. "FC_1000" means a fully connected layer with 1000 output channels. $h \times w$ is the input size while b_i is the number of FasterNet blocks at stage i. The FLOPs are calculated given the input size of 224×224.