CPUBone: Efficient Vision Backbone Design for Devices with Low Parallelization Capabilities

Supplementary Material

7. Additional Details to CPU-Efficient Design Strategies

7.1. MAC calculations

With Equation (1) we can calculate the MACs of a convolution with no bias. This further allows us to compute the reduction in MACs of the grouped MBConv variants (GrMBConv & GrFuMBConv) compared to their ungrouped counterparts (MBConv & FuMBConv), which we needed in Subsection 3.2.1. As well as calculate how the reduction of the kernel size from 3×3 to 2×2 affects MACs of the whole FuMBConv and GrFuMBConv block, which we need in Subsection 3.3. For MAC calculations, we leave out any batch normalization and activation functions of the MBConv blocks [25], due to having a minor influence on the MAC count. We further assume for simplicity, that the input channel dimension (C_{in}) and the output channel dimension (C_{out}) are the same, that the expansion factor is 4 and the stride is 1.

7.1.1. FuMBConv vs. GrFuMBConv

Following Equation (1), the MACs of the fused MBConv block (M_{fmb}) or the GrFuMBConv block can be computed as

$$\begin{aligned} M_{fmb} = & \frac{K_H \times K_W \times C_{in} \times C_{in} \times H_{out} \times W_{out}}{groups} \\ & + C_{in} \times C_{in} \times 4 \times H_{out} \times W_{out}, \end{aligned} \tag{4}$$

where the first part refers to the standard convolution and the second part to the pointwise convolution (see Figure 2). By pulling out common multiplicative factors, Equation (4) can be simplified as:

$$\begin{aligned} M_{fmb} = & (C_{in} \times C_{in} \times 4 \times H_{out} \times W_{out}) \times \\ & (K_H \times \frac{K_W}{groups} + 1), \end{aligned} \tag{5}$$

where the term of the pointwise convolution is reduced to a single number in the second bracket. The only distinction between FuMBConv [10] and GrFuMBConv in Equation (4) lies in the $groups$ parameter. Therefore, Equation (5) allows us to compute the proportion of MACs that GrFuMBConv requires relative to FuMBConv. When having a kernel of size 3×3 (and $groups = 1$ for the FuMBConv and $groups = 2$ for the GrFuMBConv, while all other parameters are the same), the MAC count of GrFuMBConv divided by FuMBConv is:

$$\begin{aligned} = & \frac{(C_{in} \times C_{in} \times 4 \times H_{out} \times W_{out})}{(C_{in} \times C_{in} \times 4 \times H_{out} \times W_{out})} \times \\ & \frac{(3 \times \frac{3}{2} + 1)}{(3 \times \frac{3}{1} + 1)}, \end{aligned} \tag{6}$$

which can be further simplified to:

$$= \frac{1}{1} \times \frac{5.5}{10} = 0.55 \quad . \tag{7}$$

Consequently, the GrFuMBConv always has 45% less MACs than its ungrouped counterpart (when they only differ in the $groups$ paramter), independent of the channel dimension.

7.1.2. MBConv vs GrMBConv

The original MBConv block [25] consists out of two pointwise convolutions and one depthwise convolution (see Figure 2). Following Equation (1), we can calculate the MAC count of the MBConv (M_{mb}) and GrMBConv block (only differing in the $groups$ parameter) as follows:

$$\begin{aligned} M_{mb} = & \frac{C_{in} \times C_{in} \times 4 \times H_{out} \times W_{out}}{groups} \\ & + C_{in} \times 4 \times K_H \times K_W \times H_{out} \times W_{out} \\ & + C_{in} \times C_{in} \times 4 \times H_{out} \times W_{out}. \end{aligned} \tag{8}$$

Similarly to what we did in Equation (5), we can simplify Equation (8) by pulling out the commmon multiplicative factors, leaving us with this:

$$\begin{aligned} M_{mb} = & C_{in} \times 4 \times H_{out} \times W_{out} \times \\ & (\frac{C_{in}}{groups} + K_H \times K_W + C_{in}). \end{aligned} \tag{9}$$

Similarly to Equation (6), we can now easily compute the relative MAC count of GrMBConv, compared to MBConv, by dividing the number of MACs of GrMBConv ($groups = 2$) with the MAC count of MBConv (with kernel size set to 3×3 for both):

$$\begin{aligned} = & \frac{C_{in} \times 4 \times H_{out} \times W_{out}}{C_{in} \times 4 \times H_{out} \times W_{out}} \times \\ & \frac{(\frac{C_{in}}{2} + 3 \times 3 + C_{in})}{(\frac{C_{in}}{1} + 3 \times 3 + C_{in})}, \end{aligned} \tag{10}$$

which can be further simplified to:

$$
\begin{aligned}
&= \frac{1}{1} \times \frac{(\frac{C_{in}}{2} + 9 + C_{in})}{(\frac{C_{in}}{1} + 9 + C_{in})} = \frac{(1.5 \times C_{in} + 9)}{(2 \times C_{in} + 9)} \\
&= \frac{1.5}{2} \times \frac{C_{in} + 6}{C_{in} + 4.5}.
\end{aligned} \tag{11}
$$

Equation (11) above shows us, that the relative MAC count of GrMBConv divided by MBConv, depends on the channel dimension. However it further shows that with increasing C_{in}, it converges to $1.5/2 = 0.75$, meaning the grouped MBConv variant has 25% less MACs than its ungrouped counterpart. For the channel dimensions in Table 1, namely 32, 64, 128, 256 and 512, the relative MACs in percentage rounded accordingly are, respectively: 78.1%, 76.6%, 75.8%, 75.4%, 75.2%. The average is 76.2%.

7.1.3. Impact of Kernel Size Reduction on MAC Count for GrFuMBConv and FuMBConv

In Subsection 3.3 we compared how a reduction of the kernel size from 3×3 to 2×2 affects MACpS for the depthwise convolution, the FuMBConv and the GrFuMBConv with $groups = 2$. In Subsection 3.1 we calculated for convolutions in general, how the reduced kernel affects MAC count, leading to approximately 56% always. Therefore the MAC count of the depthwise convolution is reduced by 56%, when reducing the kernel size. However, the FuMBConv block and GrFuMBConv block also include an additional pointwise convolution, whose kernel is not reduced, making the calculation a bit more complicated. However, we need the MAC count of the full MBConv blocks, since we also tested the full MBConv block in Subsection 3.3.

Starting from Equation (5), which describes the MAC count of the GrFuMBConv or the FuMBConv, we can divide the MACs of the 2×2 FuMBConv by the 3×3 FuMBConv block to obtain the percentage reduction (similarly to what we did in Equation (6)):

$$
\begin{aligned}
&= \frac{(C_{in} \times C_{in} \times 4 \times H_{out} \times W_{out})}{(C_{in} \times C_{in} \times 4 \times H_{out} \times W_{out})} \times \\
&\frac{2 \times \frac{2}{1} + 1}{3 \times \frac{3}{1} + 1} \\
&= \frac{1}{1} \times \frac{5}{10} = 0.5.
\end{aligned} \tag{12}
$$

Consequently, the kernel reduction does exactly halve the MAC count of the FuMBConv block. By doing the same thing now for the GrFuMBConv block, we yield:

$$
\begin{aligned}
&= \frac{(C_{in} \times C_{in} \times 4 \times H_{out} \times W_{out})}{(C_{in} \times C_{in} \times 4 \times H_{out} \times W_{out})} \times \\
&\frac{2 \times \frac{2}{2} + 1}{3 \times \frac{3}{2} + 1} \\
&= \frac{1}{1} \times \frac{3}{5.5} = 0.\overline{54} \quad ,
\end{aligned} \tag{13}
$$

meaning for the GrFuMBConv block the kernel reduction leads to approximately 46% less MACs.

7.2. Kernel Experiment GPU

In Table 8 & 9, we repeat the experiments of Subsection 3.3 from table 2 & 3, but on GPU instead of the ARM CPU of the Raspberry Pi5. Table 8 & 9 show an extreme deterioration of the MACpS, when reducing the kernel of a convolution from 3×3 to 2×2. Especially for depthwise convolutions, this change leads to a higher execution time than compared to the original 3×3 kernel. For FuMBConv and GrFuMBConv, the MACpS also decrease significantly; however, both variants retain at least a similar latency compared to the original 3×3 kernel. While they achieve roughly half the MACs, the corresponding reduction in MACpS offsets the expected efficiency gain, effectively nullifying it.

Dwise GPU		Channel Dimension				
Resol.	**Type**	128	256	512	1024	**Avg.**
7×7	*nmk*	**6.6**	**13.1**	**26.5**	**52.4**	**24.7**
	smk	1.2	2.3	4.7	9.3	4.4
14×14	*nmk*	**26.4**	**52.7**	**103.8**	**205.3**	**97.0**
	smk	4.7	9.0	18.7	36.3	17.2
28×28	*nmk*	**103.2**	**203.5**	**300.3**	**308.0**	**228.7**
	smk	18.1	35.7	71.5	68.9	48.5

Table 8. Execution efficiency experiment on GPU on the effect of kernel size reduction for depthwise convolutions, by measuring MMACs/ms. **Resol.** refers to operating resolution, **Type** refers to the kernel size ($nmk = 3 \times 3$, $smk = 2 \times 2$). Bold entries indicate whether nmk or smk variant has higher MACpS. Same as Table 2, but on GPU.

GPU		Ungrouped			Groups=2		
Resol.	**Type**	**Channel Dim.**			**Channel Dim.**		
		128	256	**Avg.**	128	256	**Avg.**
7×7	*nmk*	**661.6**	**1075.7**	**868.6**	**355.1**	**893.7**	**624.4**
	smk	284.4	882.9	583.6	145.3	627.2	386.2
14×14	*nmk*	**2374.3**	**3545.5**	**2959.9**	**1422.6**	**3026.8**	**2224.7**
	smk	1081.8	2021.3	1551.5	627.0	1671.9	1149.4
28×28	*nmk*	**4892.2**	**5385.9**	**5139.0**	**4068.5**	**4785.6**	**4427.0**
	smk	2333.9	2669.0	2501.5	2000.5	2538.2	2269.3

Table 9. Execution efficiency experiment on GPU on the effect of kernel size reduction. We test the GrFuMBConv (Groups=2) and FuMBConv (Ungrouped) block by measuring MMACs/ms. **Resol.** refers to operating resolution, **Type** refers to the kernel size ($nmk = 3 \times 3$, $smk = 2 \times 2$). Bold entries indicate whether nmk or smk variant has higher MACpS. Same as Table 3, but on GPU.

7.3. Grouping ARM CPU - additional Resolutions

Table 10 shows a similar experiment as Table 1, however we focus only on ARM CPU, but feature the additional

Grouping=2 Experiment ARM CPU Resolution	Variant	Channel Dimension 32	64	128	256	512	Avg.
7x7 (MMACs/ms)	*MBConv*	**4.1**	**8.5**	**14.9**	**22.4**	**23.5**	14.7
	***Gr**MBConv*	3.7	6.5	11.6	17.7	19.5	11.8 **(-19%)**
	***Fu**MBConv*	**25.5**	**36.3**	**38.7**	27.8	24.3	30.5
	***GrFu**MBConv*	14.7	28.3	37.2	**32.3**	**26.5**	27.8 **(-8%)**
14x14 (MMACs/ms)	*MBConv*	**7.8**	**13.3**	**21.4**	**26.4**	**26.2**	19.0
	***Gr**MBConv*	6.3	10.6	16.7	22.1	23.3	15.8 **(-16%)**
	***Fu**MBConv*	**40.6**	**45.3**	40.7	32.4	24.5	36.7
	***GrFu**MBConv*	31.0	42.1	**44.4**	**34.7**	**30.4**	36.5 **(-0%)**
28x28 (MMACs/ms)	*MBConv*	**11.7**	**18.6**	**22.5**	**26.3**	**26.9**	21.2
	***Gr**MBConv*	9.4	14.8	18.1	21.3	26.8	18.1 **(-14%)**
	***Fu**MBConv*	**53.2**	**54.6**	41.2	30.0	25.9	41.0
	***GrFu**MBConv*	45.5	53.4	**44.1**	**37.5**	**29.2**	41.9 **(+2%)**
56x56 (MMACs/ms)	*MBConv*	**8.4**	**12.8**	**19.1**	**22.7**	**30.8**	18.8
	***Gr**MBConv*	7.2	9.9	14.3	18.4	25.4	15.0 **(-20%)**
	***Fu**MBConv*	**42.3**	**37.5**	31.7	24.4	23.2	31.8
	***GrFu**MBConv*	35.3	37.3	**34.8**	**27.2**	**24.0**	31.7 **(-0%)**

Table 10. Execution efficiency of MACs, measured in MMACs/ms on the ARM CPU of the Raspberry Pi5, for the four MBConv variants (MBConv, GrMBConv, FuMBConv and GrFuMBConv) for different channel dimensions and operating resolutions. Bold entries indicate whether the grouped or ungrouped variant has higher MACpS. All grouped variants have $groups = 2$. It is similar to Table 1, but for resolutions 7×7, 14×14, 28×28 and 56×56 and only on ARM CPU.

resolutions 7×7, 28×28 and 56×56. The numbers are mostly very similar to 1, independent of the resolutions. However the GrMBConv consistently underperforms on ARM CPU, compared to the other CPU devices featured in Table 1 (Snapdragon 8 Elite CPU, Google Pixel 4 CPU). The FuMBConv block however consistently shows high MACpS, only on resolution 7×7 it falls off a bit.

7.4. Grouping 4 on ARM CPU

In Table 11 we repeat the experiment of Table 1 for additional resolutions (7×7, 14×14, 28×28 and 56×56) and on the ARM CPU, but for $groups = 4$ instead of $groups = 2$. The averaged numbers with $groups = 4$ (Table 11) are very similar to $groups = 2$ (Table 1), however the distribution over the channels is different: For channel dimensions below 128, GrFuMBConv with $groups = 4$ has 20% lower MACpS, compared to $groups = 2$. However for channel dimensions over 128, they have 36% higher MACpS. Consequently, depending on the channel dimension, either $groups = 2$ (below 128) or $groups = 4$ (over 128) is more hardware efficient.

Grouping=4 Experiment ARM CPU Resolution	Variant	Channel Dimension 32	64	128	256	512	Avg.
7x7 (MMACs/ms)	*MBConv*	**4.1**	**8.5**	**14.9**	**22.4**	**23.5**	14.7
	***Gr**MBConv*	3.1	5.4	10.2	17.4	19.9	11.2 **(-23%)**
	***Fu**MBConv*	**25.5**	**36.3**	**38.7**	27.8	24.3	30.5
	***GrFu**MBConv*	10.2	18.5	31.4	**34.9**	**32.0**	25.4 **(-16%)**
14x14 (MMACs/ms)	*MBConv*	**7.8**	**13.3**	**21.4**	**26.4**	26.2	19.0
	***Gr**MBConv*	5.6	9.2	16.1	22.1	**28.1**	16.2 **(-14%)**
	***Fu**MBConv*	**40.6**	**45.3**	40.7	32.4	24.5	36.7
	***GrFu**MBConv*	24.8	36.4	**43.2**	**41.4**	**39.8**	37.1 **(+1%)**
28x28 (MMACs/ms)	*MBConv*	**11.7**	**18.6**	**22.5**	**26.3**	26.9	21.2
	***Gr**MBConv*	8.4	13.0	17.1	23.6	**36.6**	19.7 **(-7%)**
	***Fu**MBConv*	**53.2**	**54.6**	41.2	30.0	25.9	41.0
	***GrFu**MBConv*	36.7	47.1	**49.5**	**50.5**	**45.7**	45.9 **(+12%)**
56x56 (MMACs/ms)	*MBConv*	**8.4**	**12.8**	**19.1**	**22.7**	30.8	18.8
	***Gr**MBConv*	6.3	8.9	14.2	21.6	**33.8**	17.0 **(-9%)**
	***Fu**MBConv*	**42.3**	**37.5**	31.7	24.4	23.2	31.8
	***GrFu**MBConv*	31.3	30.8	**39.7**	**43.4**	**39.5**	36.9 **(+16%)**

Table 11. Execution efficiency of MACs, measured in MMACs/ms on the ARM CPU of the Raspberry Pi5, for the four MBConv variants (MBConv, GrMBConv, FuMBConv and GrFuMBConv) for different channel dimensions and operating resolutions. Bold entries indicate whether the grouped or ungrouped variant has higher MACpS. It is similar to Table 1, but for resolutions 7×7, 14×14, 28×28 and 56×56, only on ARM CPU and all grouped variants have $groups = 4$ instead of $groups = 2$ of Table 1.

7.5. Grouping GPU - additional Resolutions

In Table 12 we repeat the experiment of Table 1 for additional resolutions (7×7, 14×14, 28×28 and 56×56) on the Nvidia TITAN RTX GPU. We observe that at lower resolutions, the grouped variants perform worse, likely due to reduced opportunities for parallelizing the convolutional operations. From Table 12 we can conclude, that grouping convolutions does not fully translate lower MAC count to a similarly low latency on GPU.

Grouping=2 Experiment GPU		Channel Dimension					
Resolution	**Variant**	32	64	128	256	512	**Avg.**
7x7 (MMACs/ms)	*MBConv*	**8.6**	**31.7**	**114.4**	**474.0**	**1746.1**	475.0
	Gr*MBConv*	5.6	20.4	82.4	328.4	1094.9	306.3 (**-35%**)
	Fu*MBConv*	**45.6**	**182.4**	**672.7**	**1089.9**	**1309.8**	660.1
	GrFu*MBConv*	23.5	85.6	342.2	878.4	1264.3	518.8 (**-21%**)
14x14 (MMACs/ms)	*MBConv*	**34.2**	**112.3**	**428.6**	**1747.3**	**3494.0**	1163.3
	Gr*MBConv*	25.0	88.4	308.8	1260.6	2679.1	872.4 (**-25%**)
	Fu*MBConv*	**185.1**	**622.6**	**2394.6**	**3612.2**	3849.0	2132.7
	GrFu*MBConv*	96.8	330.3	1307.3	2963.3	**3878.9**	1715.3 (**-19%**)
28x28 (MMACs/ms)	*MBConv*	**140.0**	**530.1**	**1865.3**	**3053.0**	**4114.5**	1940.6
	Gr*MBConv*	96.7	384.0	1350.1	2395.3	3769.6	1599.1 (**-17%**)
	Fu*MBConv*	**752.8**	**3043.4**	**4882.2**	**5472.1**	5676.0	3965.3
	GrFu*MBConv*	395.0	1452.0	4237.8	4839.3	**5703.9**	3325.6 (**-16%**)
56x56 (MMACs/ms)	*MBConv*	**524.9**	**1503.4**	**2518.9**	**3820.4**	**4717.1**	2617.0
	Gr*MBConv*	379.8	1048.2	1953.9	3171.3	4416.1	2193.8 (**-16%**)
	Fu*MBConv*	**2930.6**	**4971.6**	**6183.8**	**7226.3**	**7576.2**	5777.7
	GrFu*MBConv*	1402.0	3654.2	5068.1	6606.4	7288.1	4803.8 (**-17%**)

Table 12. Execution efficiency of MACs, measured in MMACs/ms on the Nvidia TITAN RTX GPU, for the four MBConv variants (MBConv, GrMBConv, FuMBConv and GrFuMBConv) for different channel dimensions and operating resolutions. Bold entries indicate whether the grouped or ungrouped variant has higher MACpS. All grouped variants have $groups = 2$. It is similar to Table 1, but for resolutions 7×7, 14×14, 28×28 and 56×56 and only on GPU.

References

[1] Han Cai, Junyan Li, Muyan Hu, Chuang Gan, and Song Han. Efficientvit: Lightweight multi-scale attention for high-resolution dense prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17302–17313, 2023. 1, 2, 3, 5, 6, 7

[2] Jierun Chen, Shiu-hong Kao, Hao He, Weipeng Zhuo, Song Wen, Chul-Ho Lee, and S-H Gary Chan. Run, don't walk: Chasing higher flops for faster neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12021–12031, 2023. 1, 2, 3, 6, 7

[3] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 7

[4] Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Xiaoyi Dong, Lu Yuan, and Zicheng Liu. Mobile-former: Bridging mobilenet and transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5270–5279, 2022. 1

[5] Zekai Chen, Fangtian Zhong, Qi Luo, Xiao Zhang, and Yanwei Zheng. Edgevit: Efficient visual modeling for edge computing. In *International Conference on Wireless Algorithms, Systems, and Applications*, pages 393–405. Springer, 2022. 2, 7, 8

[6] MMSegmentation Contributors. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/open-mmlab/mmsegmentation, 2020. 7

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 7

[8] Qihang Fan, Huaibo Huang, Xiaoqiang Zhou, and Ran He. Lightweight vision transformer with bidirectional interaction. *Advances in Neural Information Processing Systems*, 36, 2023. 7, 8

[9] Goutam Yelluru Gopal and Maria A Amer. Separable self and mixed attention transformers for efficient object tracking. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 6708–6717, 2024. 2

[10] Suyog Gupta and Mingxing Tan. Efficientnet-edgetpu: Creating accelerator-optimized neural networks with automl. *https://ai.googleblog.com/2019/08/efficientnetedgetpu-creating.html*, 2019. 3, 1

[11] 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. 2, 6, 7

[12] Andrew 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/CVF international conference on computer vision*, pages 1314–1324, 2019. 8

[13] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6399–6408, 2019. 7

[14] Yanyu Li, Ju Hu, Yang Wen, Georgios Evangelidis, Kamyar Salahi, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. Rethinking vision transformers for mobilenet size and speed. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16889–16900, 2023. 6, 7

[15] 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 *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014. 7

[16] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 7, 8

[17] Alejandro López-Ortiz, Alejandro Salinger, and Robert Suderman. Toward a generic hybrid cpu-gpu parallelization of divide-and-conquer algorithms. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, pages 601–610. IEEE, 2013. 2

[18] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 7

[19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 7

[20] Xu Ma, Xiyang Dai, Jianwei Yang, Bin Xiao, Yinpeng Chen, Yun Fu, and Lu Yuan. Efficient modulation for vision networks. *arXiv preprint arXiv:2403.19963*, 2024. 1, 2, 6, 7

[21] Mustafa Munir, William Avery, and Radu Marculescu. Mobilevig: Graph-based sparse attention for mobile vision applications. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 2211–2219, 2023. 6

[22] Moritz Nottebaum, Matteo Dunnhofer, and Christian Micheloni. Lowformer: Hardware efficient design for convolutional transformer backbones. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 7008–7018. IEEE, 2025. 1, 2, 3, 4, 5, 6, 7, 8

[23] Moritz Nottebaum, Matteo Dunnhofer, and Christian Micheloni. Beyond macs: Hardware efficient architecture design for vision backbones, 2026. 6

[24] Danfeng Qin, Chas Leichner, Manolis Delakis, Marco Fornoni, Shixin Luo, Fan Yang, Weijun Wang, Colby Banbury, Chengxi Ye, Berkin Akin, et al. Mobilenetv4: universal models for the mobile ecosystem. In *European Conference on Computer Vision*, pages 78–96. Springer, 2024. 1, 2, 3, 8

[25] 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. 2, 3, 8, 1

[26] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*, pages 10096–10106. PMLR, 2021. 1, 3

[27] Yehui Tang, Kai Han, Jianyuan Guo, Chang Xu, Chao Xu, and Yunhe Wang. Ghostnetv2: Enhance cheap operation with long-range attention. *Advances in Neural Information Processing Systems*, 35:9969–9982, 2022. 2, 6

[28] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff Zhu, Oncel Tuzel, and Anurag Ranjan. Fastvit: A fast hybrid vision transformer using structural reparameterization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5785–5795, 2023. 1, 2, 5, 6, 7

[29] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff Zhu, Oncel Tuzel, and Anurag Ranjan. Mobileone: An improved one millisecond mobile backbone. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7907–7917, 2023. 1, 2, 6

[30] Ao Wang, Hui Chen, Zijia Lin, Jungong Han, and Guiguang Ding. Repvit: Revisiting mobile cnn from vit perspective. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15909–15920, 2024. 1, 2, 6, 7

[31] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, et al. Yolov10: Real-time end-to-end object detection. *Advances in Neural Information Processing Systems*, 37:107984–108011, 2024. 2

[32] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pvt v2: Improved baselines with pyramid vision transformer. *Computational Visual Media*, 8(3):415–424, 2022. 7, 8

[33] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers, 2021. 1, 2

[34] Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. Metaformer is actually what you need for vision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10819–10829, 2022. 2

[35] Seokju Yun and Youngmin Ro. Shvit: Single-head vision transformer with memory efficient macro design. *arXiv preprint arXiv:2401.16456*, 2024. 2, 5

[36] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 7

[37] Lei Zhu, Xinjiang Wang, Zhanghan Ke, Wayne Zhang, and Rynson WH Lau. Biformer: Vision transformer with bi-level routing attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10323–10333, 2023. 6