MobileDets: Searching for Object Detection Architecture for Mobile Accelerators

Anonymous Author(s)
Affiliation
Address
email

Abstract

Inverted bottleneck layers, which are built upon depthwise convolutions, have been the predominant building blocks in state-of-the-art object detection models on mobile devices. In this work, we investigate the optimality of this design pattern over a broad range of mobile accelerators by revisiting the usefulness of regular convolutions. We achieve substantial improvements in the latency-accuracy trade-off by incorporating regular convolutions in the search space, effectively placing them in the network via neural architecture search, and directly optimizing the network architectures for object detection. We obtain a family of object detection models, MobileDets, that achieve state-of-the-art results across mobile accelerators. On the COCO object detection task, MobileDets outperform MobileNetV3+SSDLite by 1.7 mAP at comparable mobile CPU inference latencies. MobileDets also outperform MobileNetV2+SSDLite by 1.9 mAP on mobile CPUs, 3.7 mAP on EdgeTPUs, 3.4 mAP on DSPs and 2.7 mAP on edge GPUs without latency increase. Moreover, MobileDets are comparable with the state-of-the-art MnasFPN on mobile CPUs even without using the feature pyramid, and achieve better mAP scores on both EdgeTPUs and DSPs with up to 2× speedup.

1 Introduction

Neural architecture search (NAS) methods [3, 19, 18, 8, 6] have demonstrated a superior ability in finding models that are not only accurate but also efficient on a specific hardware platform. Despite many advancements in NAS algorithms [22, 15, 1, 14, 3, 19, 18, 8, 6], inverted bottlenecks (IBN) [16] remain the predominant building block in state-of-the-art mobile models. IBN-only search spaces have also been the go-to setup in majority of the related NAS publications [18, 3, 19, 8, 2].

Inverted bottleneck layers rely on large depthwise convolutions [17], which have relatively low FLOPS and parameter counts and can be executed efficiently on CPUs. However, the advantage of depthwise convolutions for mobile inference is less clear for hardware accelerators such as DSPs or EdgeTPUs which are becoming increasingly popular on mobile devices. For example, for certain tensor shapes and kernel dimensions, a regular convolution can run 3× as fast as the depthwise variation on an EdgeTPU despite having 7× as many FLOPs. This observation leads us to question the exclusive use of IBN-only search spaces in most current state-of-the-art mobile architectures.

Our work seeks to rethink the use of IBN-only search space on modern mobile accelerators. To this end, we propose the MobileDet search space family which includes not only IBNs but also flexible full convolution sequences motivated by the structure of tensor decomposition [20, 4]. Using the object detection task as an example (one of the most popular mobile vision applications), we show the MobileDet search space family can enable NAS methods to identify models with substantial better latency-accuracy trade-offs on mobile CPUs, DSPs, EdgeTPUs and edge GPUs.

Submitted to the 4th Workshop on Meta-Learning at NeurIPS 2020, Vancouver, Canada. Do not distribute.

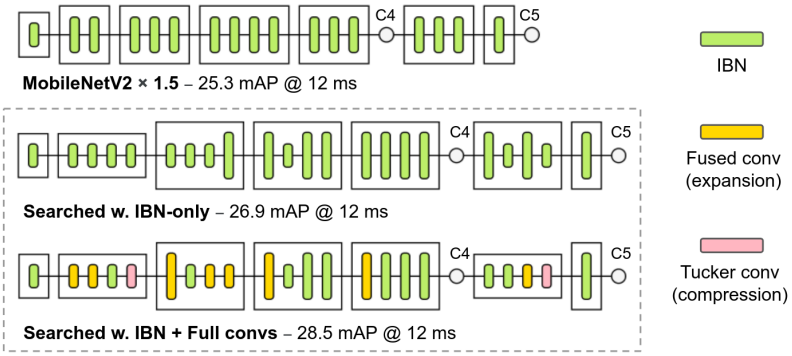

Figure 1: Platform-aware NAS and our MobileDet search space work synergistically to boost object detection performance on accelerators. SSDLite object detection performance on Pixel-4 DSPs with different backbone designs: manually-designed MobileNetV2, searched with IBN-only search space, and searched with the proposed MobileDet space (with both IBNs and full conv-based building blocks). Layers are visualized as vertical bars where color indicates layer type and length indicates expansion ratio. $C4$ and $C5$ mark the feature inputs to the SSDLite head. While conducting platform-aware NAS in an IBN-only search space achieves a 1.6 mAP boost over the handcrafted baseline, searching within the MobileDet space brings another 1.6 mAP gain.

To evaluate our proposed MobileDet search space, we perform latency-aware NAS for object detection, targeting a diverse set of mobile platforms. Experimental results show that by using our MobileDet search space family and directly searching on detection tasks, we can consistently improve the performance across all hardware platforms. Our searched models, MobileDets, outperform MobileNetV2 by 1.9mAP on CPU, 3.7mAP on EdgeTPU, 3.4mAP on DSP, and 2.7mAP on edge GPU at comparable inference latencies. MobileDets also outperform the state-of-the-art MobileNetV3 classification backbone by 1.7mAP at similar CPU inference efficiency. Further, the searched models achieved comparable performance with the state-of-the-art mobile CPU detector, MnasFPN [5], without leveraging the NAS-FPN head which may complicate the deployment. On both EdgeTPUs and DSPs, MobileDets are more accurate than MnasFPN while being more than twice as fast.

Our main contributions can be summarized as follows:

- Unlike many existing works which exclusively focused on IBN layers for mobile applications, we propose an augmented search space family with building blocks based on regular convolutions. We show that NAS methods can substantially benefit from this enlarged search space to achieve better latency-accuracy trade-off on a variety of mobile devices.
- We deliver MobileDets, a set of mobile object detection with state-of-the-art quality-latency trade-off across CPUs, EdgeTPUs, DSPs and NVIDIA Jetson Xavier GPUs. Models will be released to benefit a wide range of on-device object detection applications.

2 Revisiting Full Convs in Mobile Search Spaces

The layout of an Inverted Bottleneck (IBN) is illustrated in Figure 2. IBNs are designed to reduce the number of parameters and FLOPs, and leverage the depthwise-separable kernels to achieve high efficiency on mobile CPUs. However, not all FLOPs are the same, especially in modern mobile accelerators such as EdgeTPU and DSPs. For example, a regular convolution may run $3\times$ as fast on EdgeTPUs than its depthwise variation even with $7\times$ as many FLOPs. The observation indicates that the widely used IBN-only search space can be suboptimal for modern mobile accelerators. This motivated us to propose new building blocks by revisiting regular (full) convolutions to enrich IBN-only search spaces for mobile accelerators. Specifically, we propose two flexible layers to perform channel expansion and compression, respectively, which are detailed below.

2.1 Fused Inverted Bottleneck Layers (Expansion)

The depthwise-separable convolution [17] is a critical element of an inverted bottleneck [9] (Figure 2). The idea behind the depthwise-separable convolution is to replace an "expensive" full convolution with a combination of a depthwise convolution (for spatial dimension) and a 1×1 pointwise

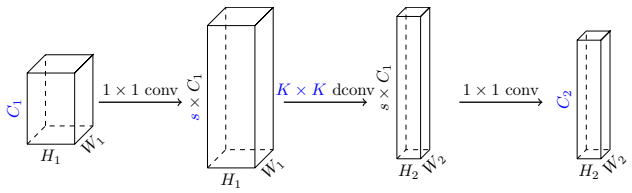

Figure 2: Inverted bottleneck layer: 1×1 pointwise convolution transforms the input channels from C_1 to $s \times C_1$ with input expansion ratio $s > 1$, then $K \times K$ *depthwise* convolution transforms the input channels from $s \times C_1$ to $s \times C_1$, and the last 1×1 pointwise convolution transforms the channels from $s \times C_1$ to C_2. The highlighted C_1, s, K, C_2 in IBN layer are searchable.

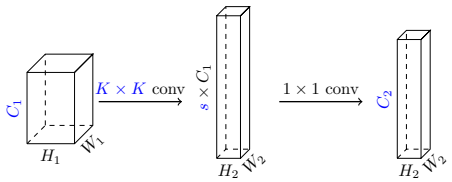

Figure 3: Fused inverted bottleneck layer: $K \times K$ *regular* convolution transforms the input channels from C_1 to $s \times C_1$ with input expansion ratio $s > 1$, and the last 1×1 pointwise convolution transforms the channels from $s \times C_1$ to C_2. The highlighted C_1, K, s, C_2 in the fused inverted bottleneck layer are searchable.

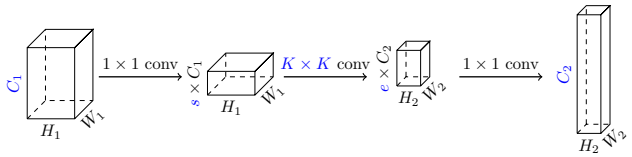

Figure 4: Tucker layer: 1×1 pointwise convolution transforms the input channels C_1 to $s \times C_1$ with input compression ratio $s < 1$, then $K \times K$ *regular* convolution transforms the input channels from $s \times C_1$ to $e \times C_2$ with output compression ratio $e < 1$, and the last 1×1 pointwise convolution transforms the channels from $e \times C_2$ to C_2. The highlighted C_1, s, K, e, C_2 in Tucker layer are searchable.

convolution (for channel dimension). However, the notion of expensiveness was largely defined based on FLOPs or the number of parameters, which are not necessarily correlated with the inference efficiency on modern mobile accelerators. To incorporate regular convolutions, we propose to modify an IBN layer by fusing together its first 1×1 convolution and its subsequent $K \times K$ depthwise convolution into a single $K \times K$ regular convolution (Figure 3). Like a standard inverted bottleneck layer, the initial convolution in our fused inverted bottleneck increases the number of filters by a factor $s > 1$. The expansion ratio of this layer will be determined by the NAS algorithm.

2.2 Tucker Convolution Layers (Compression)

Bottleneck layers were introduced in ResNet [7] to reduce the cost of large convolutions over high-dimensional feature maps. A bottleneck layer with compression ratio $s < 1$ consists of a 1×1 convolution with C_1 input filters and $s \cdot C_1$ output filters, followed by a $K \times K$ convolution with $s \cdot C_1$ output filters, followed by a 1×1 convolution with C_2 output filters. We generalize these bottlenecks (Figure 4) by allowing the initial 1×1 convolution to potentially have a different number of output filters than the $K \times K$ convolution and let the NAS algorithm to decide the best configurations.

We refer to these new building blocks as *Tucker convolution layers* because of their connections with Tucker decompositions. (See Appendix A for details.)

3 Experiments

We consider COCO dataset for our object detection experiments. We report mean average precision (mAP) for object detection and the real latency of searched models on accelerators with image size 320×320. we conduct our experiments in two stages: architecture search for optimal architecture and architecture evaluation via retraining the found architecture from scratch. We provide details for these two steps, as well as our latency benchmark protocol in Appendix A.

Search Space. The overall layout of our search space resembles that of ProxylessNAS and TuNAS. We consider three search space variants with increasing sizes: ***IBN***: The smallest search space that contains IBN layers only. Each layer may choose from kernel sizes $(3, 5)$ and expansion factors $(4, 8)$; ***IBN+Fused***: An enlarged search space that not only contains all the IBN variants above, but also Fused inverted bottleneck layers in addition with searchable kernel sizes $(3, 5)$ and expansion factors $(4, 8)$; ***IBN+Fused+Tucker***: A further enlarged search space that contains Tucker (compression) layers in addition. Each Tucker layer allows searchable input and output compression ratios within $(0.25, 0.75)$. For all search spaces variants above, we also search for the output channel sizes of each layer among the options of $(0.5, 0.625, 0.75, 1.0, 1.25, 1.5, 2.0)$ times a *base channel size* (rounded to the multiples of 8 to be more hardware-friendly). Layers in the same block share the same base channel size, though they can end up with different channel multipliers. The base channel sizes for all the blocks (from stem to head) are 32-16-32-48-96-96-160-192-192. The multipliers and base channel sizes are designed to approximately subsume several representative architectures in the literature, such as MobileNetV2 and MnasNet.

Hardware-Specific Adaptations. The aforementioned search spaces are slightly adapted depending on the target hardware. Specifically: 1, All building blocks are augmented with Squeeze-and-Excitation blocks and h-Swish activation functions (to replace ReLU-6) when targeting at CPUs. This is necessary for fair comparison with the MobileNetV3+SSDLite baseline. Both primitives are not well supported on EdgeTPUs or DSPs; 2, We exclude 5×5 convolutions from the search space when targeting at DSPs, which are known to be highly inefficient due to hardware constraints.

3.1 Search Space Ablation

With a perfect architecture search algorithm, the largest search space is guaranteed to outperform the smaller ones because it subsumes the solutions of the latter. This is not necessarily the case in practice, however, as the algorithm may end up with sub-optimal solutions especially when the search space is large [5]. A search space is considered useful if it enables NAS methods to identify *sufficiently good* architectures even if they are not optimal. In the following, we evaluate the usefulness of different search spaces by pairing them with TuNAS [2], a scalable latency-aware NAS implementation.

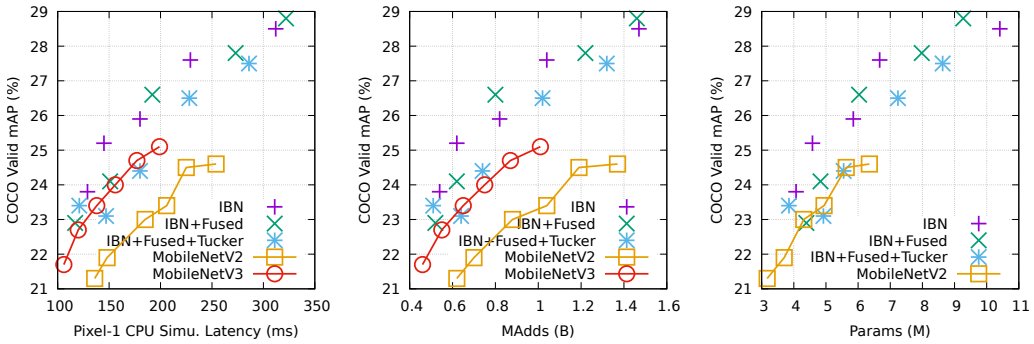

Figure 5: NAS results on Pixel-1 CPU using different search space variants.

CPU. Figure 5 shows the NAS results for Pixel-1 CPUs. As expected, MobileNetV3+SSDLite is a strong baseline as the efficiency of its backbone has been heavily optimized for the same hardware platform over the classification task on ImageNet. We also note that the presence of regular convolutions does not offer clear advantages in this particular case, as IBN-only is already strong under FLOPs/CPU latency. Nevertheless, conducting domain-specific architecture search wrt the object detection task offers non trivial gains on COCO (+1mAP in the range of 150-200ms).

EdgeTPU. Figure 6 shows the architecture search results when targeting at Pixel-4 EdgeTPUs. Conducting hardware-aware architecture search with any of the three search spaces significantly improves the overall quality. This is largely due the the fact that the baseline architecture (MobileNetV2)[1] is heavily optimized towards CPU latency, which is strongly correlated with FLOPs/MAdds but not well calibrated with the EdgeTPU latency. Notably, while IBN-only still offers the best accuracy-MAdds trade-off (middle plot), having regular convolutions in the search space (either IBN+Fused

[1]MobileNetV3 is not well supported on EdgeTPUs due to h-swish and squeeze-and-excite blocks.

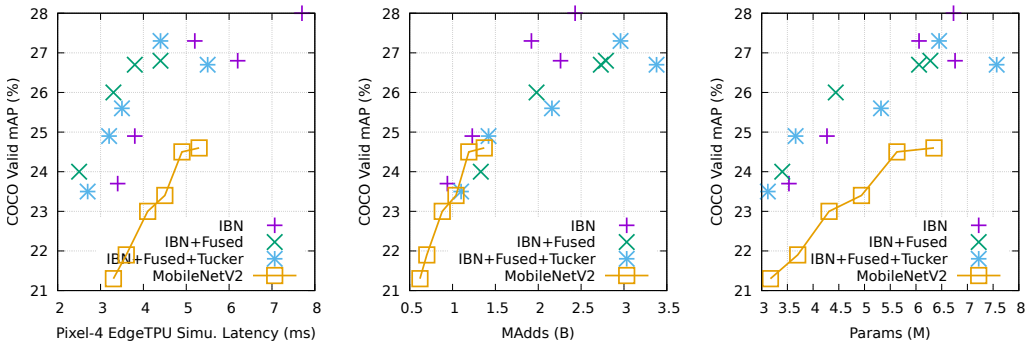

Figure 6: NAS results on Pixel-4 EdgeTPU using different search space variants.

or IBN+Fused+Tucker) offers clear further advantages in terms of accuracy-latency trade-off. The results demonstrate the usefulness of full convolutions on EdgeTPUs.

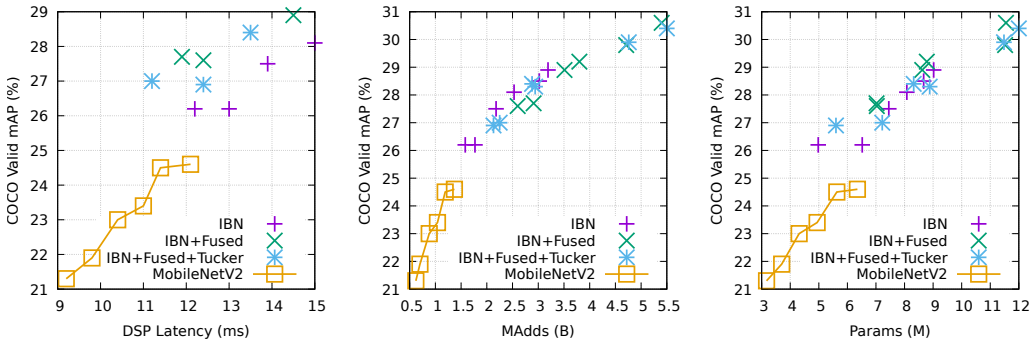

Figure 7: NAS results on Pixel-4 DSP using different search space variants.

DSP. Figure 7 shows the search results when targeting at Pixel-4 DSPs. Similar to EdgeTPUs, it is evident that the inclusion of regular convolutions in the search space leads to substantial mAP improvement under comparable inference latency.

Model/Search Space	Target hardware	mAP (%) Valid	mAP (%) Test	Latency (ms) CPU	Latency (ms) EdgeTPU	Latency (ms) DSP	MAdds (B)	Params (M)
MobileNetV2$^{\diamond\ddagger}$		–	22.1	162	8.4	11.3	0.80	4.3
MobileNetV2 (ours)$^{\diamond}$		22.2	21.8	129	6.5	9.2	0.62	3.17
MobileNetV2 $\times$ 1.5 (ours)$^{\diamond}$		25.7	25.3	225	9.0	12.1	1.37	6.35
MobileNetV3†‡		–	22.0	119	$*$	$*$	0.51	3.22
MobileNetV3 (ours)†		22.2	21.8	108	$*$	$*$	0.46	4.03
MobileNetV3 $\times$ 1.2 (ours)†		23.6	23.1	138	$*$	$*$	0.65	5.58
MnasFPN (ours)$^{\triangleleft}$		25.6	26.1	185	18.5	25.1	0.92	2.50
MnasFPN (ours)$\times$ 0.7$^{\triangleleft}$		24.3	23.8	120	16.4	23.4	0.53	1.29
IBN+Fused+Tucker†	CPU	24.2	23.7	122	$*$	$*$	0.51	3.85
IBN+Fused†	CPU	23.0	22.7	107	$*$	$*$	0.39	3.57
IBN†	CPU	23.9	23.4	113	$*$	$*$	0.45	4.21
IBN+Fused+Tucker	EdgeTPU	25.7	25.5	248	6.9	10.8	1.53	4.20
IBN+Fused	EdgeTPU	26.0	25.4	272	6.8	9.9	1.76	4.79
IBN	EdgeTPU	25.1	24.7	185	7.4	10.4	0.97	4.17
IBN+Fused+Tucker$^{\diamond}$	DSP	28.9	28.5	420	8.6	12.3	2.82	7.16
IBN+Fused$^{\diamond}$	DSP	29.1	28.5	469	8.6	11.9	3.22	9.15
IBN$^{\diamond}$	DSP	27.3	26.9	259	8.7	12.2	1.43	4.85

Table 1: Main results. Test AP scores are based on COCO test-dev. †: Augmented with Squeeze-Excite and h-Swish (CPU-only); $*$: Not well supported by the hardware platform; $\diamond$: 3$\times$3 kernel size only (DSP-friendly); $^{\triangleleft}$: Augmented with NAS-FPN head; ‡: Endpoint C4 located after the 1$\times$1 expansion in IBN.

Model	mAP (%) Valid	Test	Jetson Xavier FP16 Latency (ms)
MobileNetV2 (ours)	22.2	21.8	2.6
MobileNetV2 $\times$ 1.5 (ours)	25.7	25.3	3.4
Searched (IBN+Fused+Tucker)	27.6	28.0	3.2

Table 2: Generalization of the MobileDet search space to NVIDIA Jetson GPUs. We follow the DSP setup in Table 1, except that the number of filters are rounded to the multiples of 16.

3.2 Main Results

We compare our architectures obtained via latency-aware NAS against state-of-the-arts mobile detection models on COCO [13]. For each target hardware platform, we report results obtained by searching over each of the three search space variants. Results are presented in Table 1. On CPUs, we find that searching directly on a detection task allows us to improve mAP by 1.7mAP over MobileNetV3, but do not find evidence that our proposed fused IBNs and Tucker bottlenecks are required to obtain high quality models. On DSPs and EdgeTPUs, however, we find that our new primitives allow us to significantly improve the tradeoff between model speed and accuracy.

On mobile CPUs, MobileDet outperforms MobileNetV3+SSDLite [8], a strong baseline based on the state-of-the-art image classification backbone, by 1.7 mAP at comparable latency. The result confirms the effectiveness of detection-specific NAS. The models also achieved competitive results with MnasFPN, the state-of-the-art detector for mobile CPUs, without leveraging the NAS-FPN head which may complicate the deployment process. It is also interesting to note that the incorporation of full convolutions is quality-neutral over mobile CPUs, indicating that IBNs are indeed promising building blocks for this particular hardware platform.

On EdgeTPUs, MobileDet outperforms MobileNetV2+SSDLite by 3.7 mAP on COCO test-dev at comparable latency. We attribute the gains to both task-specific search (wrt COCO and EdgeTPU) and the presence of full convolutions. Specifically, IBN+Fused+Tucker leads to 0.8 mAP improvement together with 7% latency reduction as compared to the IBN-only search space.

On DSPs, MobileDet achieves 28.5 mAP on COCO with 12.3 ms latency, outperforming MobileNetV2+SSDLite ($\times$1.5) by 3.2 mAP at comparable latencies. The same model also outperforms MnasFPN by 2.4 mAP with more than 2$\times$ more speed-up. Again it is worth noticing that including full convolutions in the search space clearly improved the architecture search results from 26.9 mAP @ 12.2 ms to 28.5 mAP @ 11.9ms. Fig. 9 in Appendix A.4 illustrates our searched object detection architectures, MobileDets, by targeting at different mobile hardware platforms using our largest search space. One interesting observation is that MobileDets use regular convolutions extensively on EdgeTPU and DSP, especially in the early stage of the network where depthwise convolutions tend to be less efficient. The results confirm that IBN-only search space is not optimal for these accelerators.

Generalization to New Hardware The proposed search space family (IBN+Fused+Tucker) was specifically developed wrt CPUs, EdgeTPUs and DSPs. It remains an interesting question whether the search space can extrapolate to new hardware. To answer this, we provide architecture search results wrt NVIDIA Jetson GPUs (as a "holdout" device) in Table 2. The searched model within our expanded search space achieves +3.7mAP boost over MobileNetV2 while being faster. The results confirm that the proposed search space family is generic enough to handle a new hardware platform.

4 Conclusion

In this work, we question the predominant design pattern of using depthwise inverted bottlenecks as the only building block for computer vision models on the edge. Using the object detection task as a case study, we revisit the usefulness of regular convolutions over a variety of mobile accelerators, including mobile CPUs, EdgeTPUs and DSPs. Our results reveal that full convolutions can substantially improve the accuracy-latency trade-off on several accelerators when placed at the right positions in the network, which can be efficiently identified via neural architecture search. The resulting architectures, MobileDets, achieve superior detection results over a variety of hardware platforms, significantly outperforming the prior art by a large margin.

References

[1] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pages 550–559, 2018.

[2] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V Le. Can weight sharing outperform random architecture search? an investigation with tunas. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14323–14332, 2020.

[3] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[4] J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35(3):283–319, 1970.

[5] Bo Chen, Golnaz Ghiasi, Hanxiao Liu, Tsung-Yi Lin, Dmitry Kalenichenko, Hartwig Adams, and Quoc V Le. MnasFPN: Learning latency-aware pyramid architecture for object detection on mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[6] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11398–11407, 2019.

[7] 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.

[8] 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 International Conference on Computer Vision*, pages 1314–1324, 2019.

[9] 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.

[10] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[11] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[12] 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.

[13] 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.

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

[15] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[16] 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.

[17] Laurent Sifre. Rigid-motion scattering for image classification. *Ph.D. thesis section 6.2*, 2014.

[18] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[19] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[20] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[21] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision*, pages 285–300, 2018.

[22] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.