applied sciences

MDPI

Article

Hardware-Aware Mobile Building Block Evaluation for Computer Vision

Maxim Bonnaerens [1,*], Matthias Freiberger [1], Marian Verhelst [2] and Joni Dambre [1]

[1] IDLab-AIRO, Ghent University, imec, 9052 Ghent, Belgium
[2] MICAS-ESAT, KU Leuven, 3001 Leuven, Belgium
[*] Correspondence: maxim.bonnaerens@ugent.be

Abstract: In this paper, we propose a methodology to accurately evaluate and compare the performance of efficient neural network building blocks for computer vision in a hardware-aware manner. Our comparison uses pareto fronts based on randomly sampled networks from a design space to capture the underlying accuracy/complexity trade-offs. We show that our approach enables matching of information obtained by previous comparison paradigms, but provides more insights into the relationship between hardware cost and accuracy. We use our methodology to analyze different building blocks and evaluate their performance on a range of embedded hardware platforms. This highlights the importance of benchmarking building blocks as a preselection step in the design process of a neural network. We show that choosing the right building block can speed up inference by up to a factor of two on specific hardware ML accelerators.

Keywords: hardware-aware deep learning; edge computing

check for updates

Citation: Bonnaerens, M.; Freiberger, M.; Verhelst, M.; Dambre, J. Hardware-Aware Mobile Building Block Evaluation for Computer Vision. *Appl. Sci.* **2022**, *12*, 12615. https://doi.org/10.3390/app122412615

Academic Editors: Alessandro Bruno, Deliang Fan and Zhezhi He

Received: 8 November 2022
Accepted: 7 December 2022
Published: 9 December 2022

1. Introduction

In recent years, machine learning models have been widely adopted in edge devices. Applying these models to resource-constrained hardware has accelerated research into the design of both computationally efficient neural network models and specialized hardware accelerators that are optimized to execute these neural networks. Many compact neural networks for mobile deployment have been designed, but optimal performance for a given hardware platform can only be achieved when the network is co-designed for it—not every model design choice that aims to improve efficiency does so on every hardware platform.

However, designing neural networks that can run on a certain target platform with a target latency is a complex task as there are many design choices, ranging from the type of layers used to the spatial resolution and the channel width of each block. Early work was carried out by manually designing efficient networks. MobileNetV1 [1] and its introduction of depthwise separable convolutions stimulated the beginning of a research field focused on the design of efficient vision models that can be deployed on embedded and mobile platforms. Subsequently, improved model architectures were released: MobileNetV2 [2] adds inverted residuals and inverted bottlenecks and MobileNetV3 [3] expands this further by adding a squeeze-and-excitation module [4] to the bottleneck structure. Another family of efficient models based on group convolutions and a shuffle operator are the ShuffleNets [5,6]. All these models use multi-layer *building blocks* that are repeated multiple times in the network. Their design process focuses mainly on optimising the layer structure of the building blocks to make them hardware efficient. Different variations of a network are typically manually designed [6–8], or scaled using a single complexity parameter to trade-off between task performance (e.g., accuracy) and computational cost, e.g., input resolution or channel width [1,6].

In order to compare such efficient models, most studies have used curve estimates that show the trade-off between accuracy and a model complexity metric, such as FLOPs or latency, for predefined model variants of a model family. These curves allow practitioners to gain insight into these trade-offs and make the best choices for their design constraints. The problem with curve-based comparisons is that they only use a handful of model variants, while each model type still has a very large number of hyperparameters. Once the initial mobile building blocks had been established, the design process, therefore, shifted towards neural architecture search to find new state-of-the-art efficient models. Networks optimized with NAS have repeatedly been shown to outperform manually designed networks [9–13]. As efficient models are targeted towards mobile and embedded platforms, they are often resource-constrained. Hardware-aware NAS incorporates these constraints into the search, which enables the finding of optimal models for specific target platforms [14–17]. However, the effectiveness of NAS comes with the very large computational cost of training and evaluating many candidate networks. A truly wide search is only feasible when large budgets can be spent on computing resources. For this reason, in practice, the search space of current NAS methods is usually constrained using techniques such as parameter sharing [13]. These so called one-shot NAS approaches amortize the training cost of all networks in the search space by training all possible architectures together in a large supernet, reducing the cost of NAS by several orders of magnitude [18]. As a result most approaches in NAS now use a search space with a single fixed building block while searching for other important parameters, such as network depth and width or kernel filter size. The choice of the layer structure within the building block itself is mostly based on the literature, common practice or previous experience.

As we will show, the relative efficiency of network families that use different building blocks depends on both the architecture of the targeted hardware platform and on the desired accuracy range. This means that a true hardware-aware network design approach requires a hardware-aware building block selection method as a front-end before finalizing the full network architecture, either through manual design or NAS. To provide a more systematic approach to designing design spaces, [19] introduced a comparison paradigm based on empirically sampled distribution estimates. While the authors showed in their follow up work [20] that it can be used to optimize a design space, it does not consider the trade-off between complexity within a model family and hardware cost. As we will confirm in this paper, this trade-off offers crucial information, since the best choice can change, depending on the desired accuracy level, or when choosing a more powerful version of a given embedded platform type.

In this paper, we propose a methodology to evaluate and compare the performance of efficient network building blocks for computer vision in a hardware-aware manner. As in [20], which until now was the main paradigm to compare design spaces, we use a sampled approach to pre-estimate how well a model family will perform. However, instead of focusing only on the accuracy of a model family, we propose an extension based on a sampled estimate of the pareto front, highlighting the trade-off between complexity and accuracy. In Section 2, we revisit the work of [20]. Our approach is presented in Section 3, where we also demonstrate why information about the accuracy-complexity trade-off is necessary to make the best choices. We show that our extension enables matching of the information obtained by [20], but is better suited for analysis of the relationship between hardware cost and accuracy.

Finally, in Section 4, we use our methodology to analyze and compare some of the most common building block choices on various hardware platforms. Our analysis shows that, while certain building blocks constructed with depthwise separable convolutions, inverted bottlenecks and squeeze-and-excitation modules may be more efficient in terms of FLOPs, they are often not optimal for embedded ML hardware platforms when measuring the actual latency. As such, our approach can be used as a truly hardware-aware efficient building block selection step for the construction of mobile design spaces.

2. Empirical Distribution Functions

The authors of [19] use the concepts of design spaces and model families. A *model family* is a collection of related neural network models that share some set of high-level architectural design principles. These can range from very high level principles, such as convolutional neural networks versus vision transformer families, to very specific principles, such as which specific layers to use and the relation between their depth, width and input resolution in EfficientNets.

A *design space* is a concrete set of architectures that can be instantiated from a model family. A design space consists of two components: a parameterization of a model family, such that specifying a set of model hyperparameters fully defines a network instantiation, and a set of allowable values for each hyperparameter.

To make a robust analysis of a model family over a wide range of network hyperparameters, Radosavovic et al. [20] introduced a new methodology—comparing model families using empirical distribution estimates. Instead of comparing handpicked variants of a network, they sample from a *design space* that parameterizes a network family over a wide range of network parameters and train the variants to collect accuracy metrics, which can be compared using empirical distribution functions (EDF). The normalized error EDF of a design space with n models with errors e_i is given by:

$$\hat{F}(e) = \frac{1}{n}\sum_{i=1}^{n} w_i \mathbf{1}[e_i < e] \tag{1}$$

$\hat{F}(e)$ gives the fraction of models with error less than e, with normalizing factor w to control for model complexity.

While this approach enables comparison of entire model families and has been shown to be a valuable tool to create better design spaces [20], the metric it uses does not capture differences in accuracy/complexity trade-offs between different model families. This makes EDFs a useful tool when designing model architectures around a certain predefined complexity target.

3. Capturing Trade-Offs and Search Space Difficulty

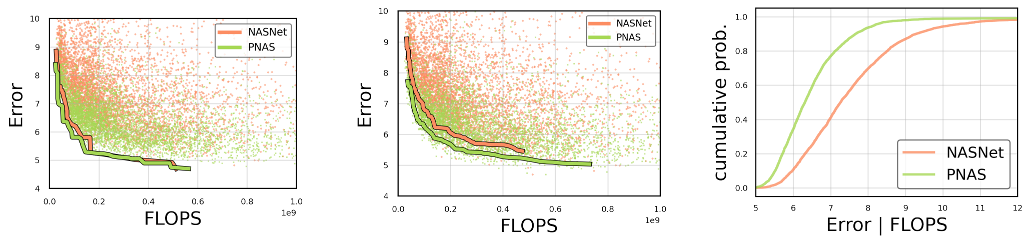
3.1. Random Sampling versus Best Models

The aim of hardware-aware network design is to optimally use the available computing budget to find solutions that are as close as possible to the accuracy/complexity pareto front for the targeted hardware platform. Since, in practice, there is a bound on the total number of network variants that can be evaluated within a reasonable time, it is beneficial to start from a family with similar *best* networks, but a higher density of *good* networks.

In [19], Radosavovic et al. demonstrated their paradigm of comparing EDFs on different model families by generating the NDS dataset, which consists of 5000 models from each corresponding design space. In Figure 1, we show 5000 trained models from two of the network families in this dataset, the NASNet [21] and the PNAS [11] model families, and highlight the best of those models on an accuracy/complexity curve.

Comparing only the best models from these larger sampling spaces, as depicted in Figure 1a, indicates that both families are capable of achieving similar results. However, when we look at all the models in the scatterplots and the corresponding EDF in Figure 1c, we can see that, when randomly sampled, the PNAS family produces more models with good accuracy than the NASNet family. This means that, for a model designer or NAS-method, it is easier to find a well-performing model for PNAS than for NASNet.

Instead of trying to estimate the true pareto front, we propose to use a random sampling accuracy/complexity curve based on a much lower number of samples (only 130). As can be seen in Figure 1b, the quality of this approximation decreases more rapidly for the NASNet family. This means that, like the approach in [19], it implicitly captures the underlying difficulty of finding good solutions, but, as an added value, this approach maintains the trade-off information.



(**a**) (**b**) (**c**)

Figure 1. (**a**) Comparing the best performing models out of 5000 generated ones for the PNAS and NASNet families shows that they can achieve similar accuracies, but, in general, PNAS will provide good models much faster, as can be seen in the normalized error EDF (**c**), which is also implicitly captured when creating accuracy/complexity curves (**b**) through random sampling with only 130 samples.

In essence, our methodology to evaluate model families involves executing the following steps: (1) create a design space for the model family to be evaluated; (2) randomly sample $\pm$130 models from this design space and train them in a low epoch regime; (3) collect hardware costs, such as latency from the target hardware platforms; and (4) create a pareto curve based on the optimal accuracy/complexity points from the sampled models.

3.2. Sample Size

Training multiple neural networks to evaluate a model family is an expensive step in the process of designing a new model. While it is not necessary to train every sampled model until convergence, as a low-epoch regime suffices to gather insights [20], we still aim to minimize the required number of samples.

We therefore analyzed the effect of the sampling size on the complexity/accuracy curve. To quantify this, we measured the average standard deviation across the accuracy/complexity curve over 100 repetitions of a certain sample size. To determine the best point in this cost-benefit trade-off, we use the Kneedle algorithm [22] to find the elbow point where the cost to train extra networks is no longer worth the expected decrease in noise on the pareto curve. Figure 2 shows the trendlines of the average standard deviation across the accuracy/FLOPs curve for increasing sample sizes, as well as the elbow points. For all families in the NDS dataset, this elbow point occurs for a sample size between 80 and 180, with a mean of 128. Similarly for the accuracy/complexity curves, with number of parameters and number of activations as complexity metrics, the mean elbow point lies at 105 and 117 samples. We, therefore, recommend a sample size of $\pm$130 when using random sampling pareto curves.

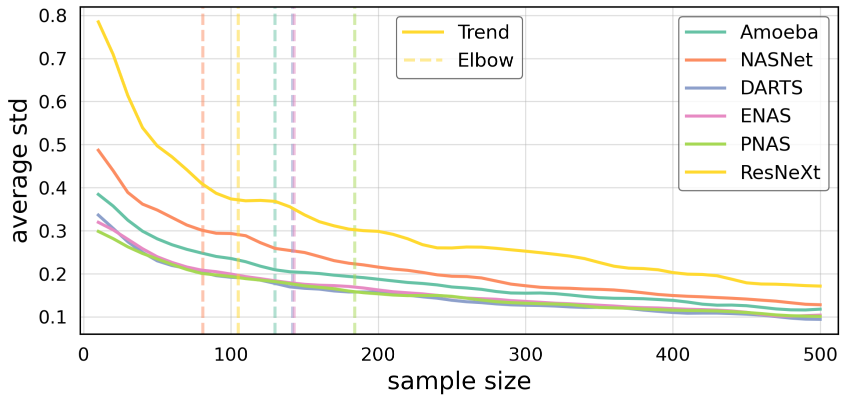


Figure 2. Trendlines of the average standard deviation on the accuracy/FLOPs curve for increasing sample size, together with their elbow points where the extra cost to train extra networks is not worth it.

3.3. Comparison to Empirical Distribution Functions

In this section, we illustrate why it is beneficial to keep information about the accuracy/complexity trade-off.

We first revisit the results from [19] for the DARTS [10] and ResNeXt [23] families to compare them to our approach. The corresponding normalized EDFs can be seen in Figure 3 (top row), where "Error | Complexity metric" denotes the complexity metric that is used to normalize the EDF. To make a fair comparison, we only use 130 sampled models for both approaches ([19] have also shown that any sample size above 100 provides a reasonable estimation). Using their approach, Radosavovic et al. concluded that the DARTS and ResNeXt models are similar when normalized by the total number of trainable parameters and that the DARTS family is a better choice than the ResNeXt models when normalized by flops (top left and middle plots). The main insight obtained from these EDFs is that they show which family provides more higher performing models. However, they cannot show how this is related to the complexity of the models.

In the bottom plots of Figure 3, we present our random sampling accuracy/complexity curves. It can be seen that the ResNeXt and DARTS families perform similarly in the FLOPs domain until 150M FLOPs, where DARTS models start to outperform the ResNeXt models. In the parameter domain, we can see that the ResNeXt family produces slightly more efficient models in the low parameter setting, but the DARTS family can reduce the error with larger models. Using EDFs provides less insight as the difference in the complexity distribution between model families becomes larger. When comparing the same ResNeXt and DARTS models based on their number of activations, which is a better proxy than FLOPs or parameters for certain memory-bound hardware accelerators, such as GPUs and TPUs [9], the corresponding EDF still shows that the ResNeXt and DARTS models perform similarly, while our performance curve shows that the ResNeXt family outperforms DARTS in the lower complexity domain by a significant margin.

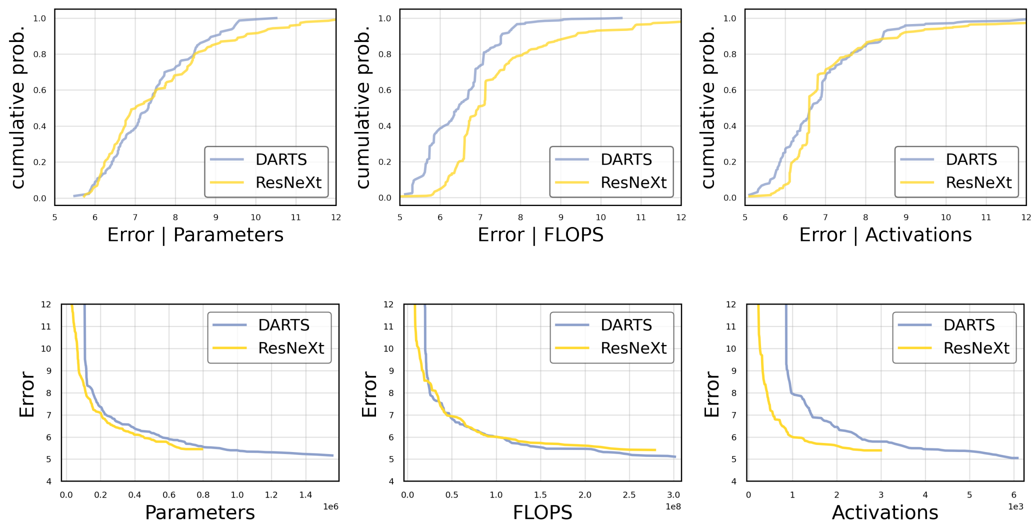


Figure 3. (**top**) The normalized error EDFs highlight that DARTS models produce more highly accurate models. (**bottom**) Random sampling accuracy/complexity curves show that, in the low complexity domain, ResNeXt outperforms DARTs.

4. Hardware-Aware Mobile Building Blocks Evaluation

In this section, we present the results of our evaluation of mobile building blocks for convolutional neural networks on various hardware platforms.

4.1. Mobile Convolutional Building Blocks

Our focus is on comparing model families that are defined only by their convolutional *building block*, i.e., all models in the family have a structure that consists of a sequence of repeated blocks, which can have different structural parameters, such as input resolution and output channels, but share the same layer structure. The different model families that we compare use the same *design space* and only differ in the used building block.

In recent years, one building block has been dominant in hardware-aware designed neural networks [3,15,24,25]—the mobile inverted bottleneck convolution (MBConv). The structure of the building block is illustrated in Figure 4. It consists of a depthwise separable convolution used in an inverse bottleneck structure and, in many cases, a squeeze-and-excitation unit.

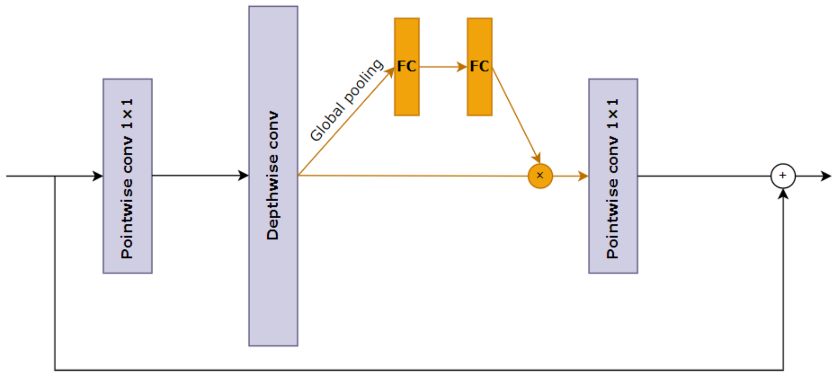


Figure 4. Structure of the MBConv building block, consisting of a depthwise separable convolution with an inverse bottleneck and squeeze-and-excitation unit.

While it has been shown to be a computationally efficient building block, recent work has shown that the large variation within layers of edge models due to the use of various diverse compression techniques results in throughput and energy efficiency shortcomings [26].

In the remainder of this article, we evaluate alternative building blocks which differ in the convolutional layer type, the bottleneck structure and the inclusion of a squeeze-and-excitation unit.

4.2. Setup

We evaluate building blocks for mobile vision models based on the *RegNet* design space [20]. This is defined by four parameters: depth d, initial width w_0, slope w_a and quantization w_m. Instead of making the width of every block in the neural network a hyperparameter, the RegNet design space parameterizes the block widths as $u_j = w_0 + w_a \cdot j$ for $0 \le j < d$. This block width is additionally quantized through the hyperparameter w_m, such that the network only increases width at each of the four stages, where each stage consists of a sequence of identical blocks (see [20] for more details).

For all evaluations, we sampled 130 models (as described in Section 3.2) from the design spaces created by the evaluated building block and trained each model for 10 epochs. The models were trained using SGD and a cosine learning rate schedule with initial learning rate 0.05, a weight decay of 10^{-4} and a batch size of 128 on an RTX 3090.

All evaluated networks were trained on the Visual Wake Words dataset [27]. This is specifically designed for vision models in embedded applications. It represents the vision use-case of identifying whether a person is present in an image or not. The dataset was derived from the publicly available COCO dataset and provides a realistic benchmark for tiny vision models.

4.3. Hardware Platforms

We evaluated the different mobile building block choices on a range of hardware platforms. The platforms evaluated were a CPU and GPU found in modern mobile phones, such as the Pixel 4, a dedicated hardware accelerator for edge devices (i.e., the Intel Movidius Myriad X Vision Processing Unit (VPU)), and an embedded GPU from NVIDIA found on the Jetson Nano. We also included a benchmark on a server-grade GPU to highlight the difference between choices for mobile deployment versus cloud deployment. Table 1 gives an overview of the hardware platforms and also includes the inference framework used as it also influences the inference speed.

The latencies from the mobile CPU, mobile GPU and VPU were obtained using nn-Meter [28], a highly accurate latency prediction library.

Table 1. Evaluated hardware platforms.

	Device	**Processor**	**Framework**
mobile CPU	Pixel4	CortexA76 CPU	TFLite
mobile GPU	Pixel4	Adreno 640 GPU	TFLite
VPU	Intel NCS2	MyriadX VPU	OpenVINO
embedded GPU	Jetson Nano	NVIDIA Maxwell GPU	TensorRT
server GPU	Server	NVIDIA RTX 3090 GPU	PyTorch

4.4. Depthwise Separable Convolutions versus Standard Convolutions

Since its introduction in MobileNet [1], depthwise separable convolutions have been the de facto building block for efficient vision models. A depthwise separable convolution factorizes a standard convolution into a depthwise convolution and a 1×1 convolution called a pointwise convolution. This factorisation reduces the computations by a factor of $\frac{1}{C_{out}} + \frac{1}{K^2}$, where C_{out} is the number of output channels of the convolution and K the kernel size. For the commonly used 3×3 kernel size, this means a reduction by a factor of eight to nine. While depthwise separable convolutions were initially introduced for small vision models, they became standard in vision models across all complexity ranges, for example, the very large vision model EfficientNet-L2 with 480M parameters, and 585B FLOPS also uses them.

Although standard convolutions may be more computationally expensive, they can, in certain cases, utilize the hardware resources more effectively. Hardware accelerators for DL commonly use wide single-instruction multiple-data (SIMD) processing units to achieve a high FLOP/S throughput. However high throughput can only be achieved if there is enough data re-use to fully utilize the processing units. In depthwise convolutions, the data re-use is much lower than in standard convolutions as each input feature map is only used in the computation of its corresponding output feature map. During training, resource utilization can be improved by using large batch sizes, obtaining data re-use with the convolutional kernel weights. However, during inference, where, typically, batch sizes of one are used, this places a memory bottleneck on the hardware accelerator. As a result, fused versions of the MobileNetV2 building block (Fused-MBConv), where the first pointwise and depthwise convolution are fused into a standard convolution, have recently been included in some hardware-aware neural architecture searches [24,29,30] and been shown to be amongst the best models on certain hardware platforms.

Figure 5 compares the randomly sampled pareto front of the design space based on the depthwise separable convolution to the design space based on the standard convolution. It can be seen that depthwise separable convolutions outperform standard convolutions when comparing FLOPs. The performance curves for the mobile CPU and mobile GPU have similar trend lines and also favour the depthwise separable convolutions. For the Jetson Nano embedded GPU, however, we see that the theoretical advantage of standard convolutions no longer translates into faster latency. Instead, standard convolutions outperform depthwise convolutions as the complexity grows. On the VPU and server-grade GPU, the initial order is reversed and standard convolution executes faster than the depthwise separable convolution.

To quantify the difference, we selected models with equal FLOPs from both families and compared the latencies on the different hardware platforms. On the mobile CPU, models from both families with equal FLOPs also have about equal latencies, but, on the VPU, the inference time of the depthwise separable convolution models is twice as long as for the standard convolution models. A similar trend was seen on the embedded GPU, where the latencies of those models were a factor of 1.85× longer.

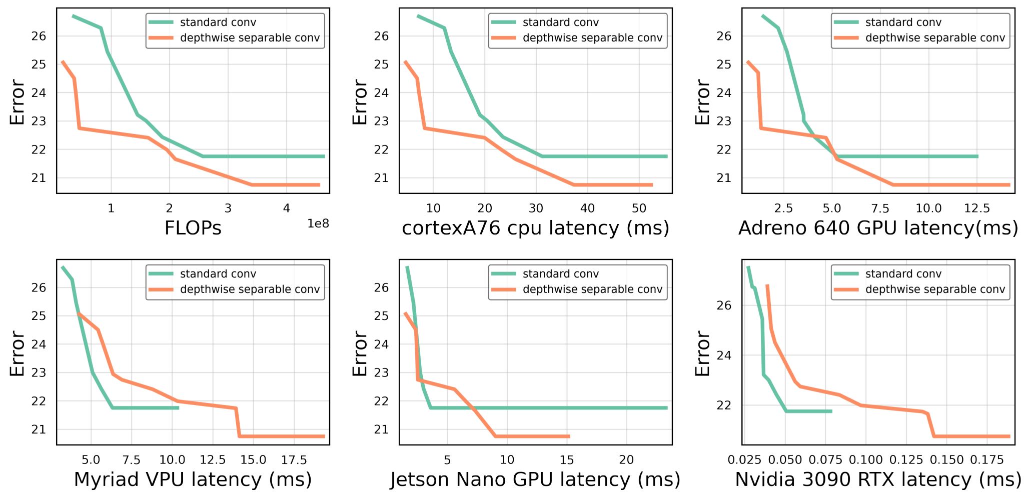


Figure 5. Depthwise separable convolutions are efficient when evaluated by FLOPS (upper left), but this does not translate to faster inference on all embedded hardware platforms.

4.5. Grouped Convolutions

In grouped convolutions, the input channels are divided into multiple groups over which a normal convolution is performed. This puts their cost in FLOPs between those of depthwise separable and standard convolutions. This is confirmed in Figure 6 (top left). Depthwise convolutions can be considered as a special case of grouped convolutions, where the number of groups equals the number of input and output channels. Similarly, standard convolutions are a special case where there is only one group.

In theory, grouped convolutions should be able to improve hardware utilization through their improved data reuse, while still using significantly fewer FLOPs than standard convolutions. However, current implementations in most deep learning frameworks fail to leverage these advantages, which in practice makes grouped convolutions slower than their standard convolution counterpart [31]. Figure 6 shows that grouped convolutions are never the most promising solution on any of the tested hardware platforms. This means that, for most target latency ranges and platforms, using the much larger search space offered by grouped convolutions will rarely be beneficial. Instead, it is more efficient to narrow down the model design space to either depthwise separable convolutions or standard convolutions, based on their relative performance in trade-off plots like that shown in Figure 6.

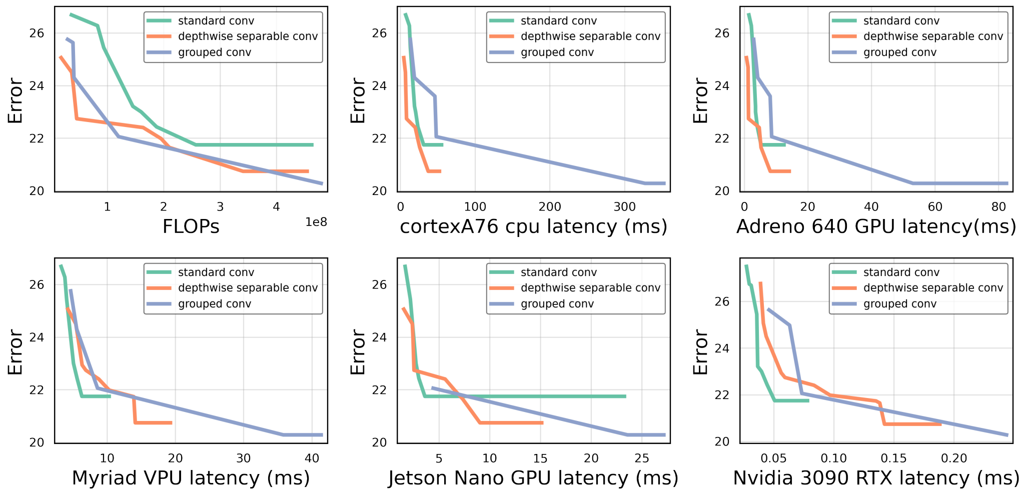


Figure 6. While grouped convolutions should, in theory, be able to combine advantages from depthwise and standard convolutions, current implementation across various hardware accelerators and inference platforms fail to leverage these advantages, making grouped convolutions not an optimal choice.

4.6. Bottleneck versus Inverted Bottleneck

Bottlenecks were introduced, together with residual connections, in ResNets [8]. The main idea behind this was to lower the computational cost of each building block by performing the convolutions at a lower dimension, such that more building blocks can be stacked and deeper networks can be created without significantly increasing the computational cost.

A standard bottleneck structure consists of three convolutional layers: a 1×1 pointwise convolution to reduce the channel size, a standard convolution to improve the features, and a final 1×1 pointwise convolution to restore the channel dimensions. A parallel residual path connects the input and output of this block. In MobileNetV2 [2], an *inverted* bottleneck was introduced where the residual connection is moved to connect the bottlenecks. In practice, this means that an inverted bottleneck consists of a pointwise convolution to expand, instead of reduce, the channel dimension, a (depthwise) convolution and a final pointwise convolution to restore the original channel dimension. The motivation for this design was that it is more hardware efficient as only the bottleneck lower dimension tensors need to be fully saved to memory as the intermediate higher channel dimension tensors are only used in depthwise convolutions. This means the tensor can be split into smaller ones.

A third possibility is to use no bottleneck, since the RegNet paper found that their best models used a bottleneck expansion/reduction factor of 1.0 [20]. When using no bottleneck, we also drop the final pointwise convolution present in bottleneck structures, since it is no longer required to make channel sizes match. All our building blocks in this comparison make use of depthwise convolutions.

The results, as shown in Figure 7, indicate that using inverted bottlenecks degrades performance and fails to deliver the promised expected inference speedup. In general, across all hardware platforms, the blocks without a bottleneck and with a regular bottleneck show very similar performance. It can be observed that the VPU and server-grade GPU slightly favour the building block without a bottleneck as it executes faster compared to bottleneck blocks. These are the same hardware platforms as those where the standard convolution outperformed the depthwise separable convolution, indicating that pointwise convolutions underperform on these platforms.

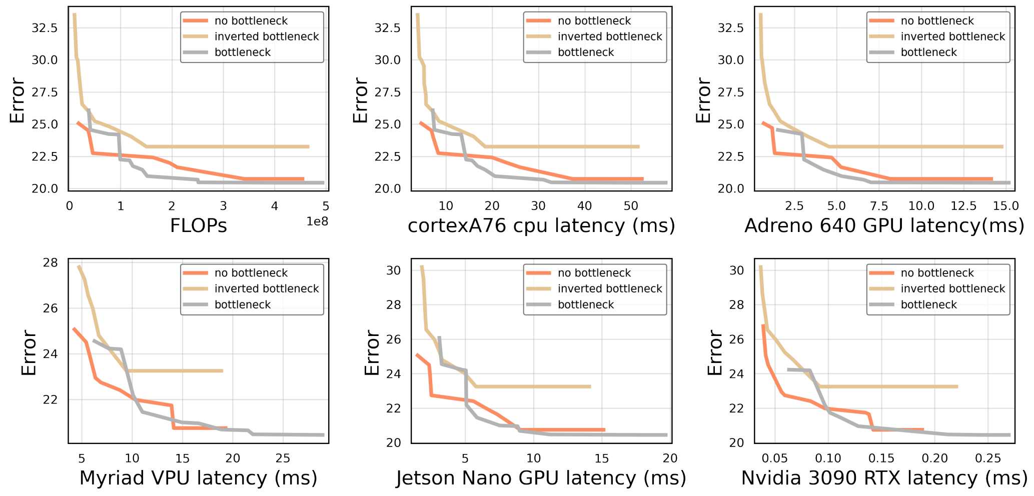


Figure 7. The bottleneck structure has little influence on the latency across all tested hardware platforms. The accuracy of inverted bottlenecks significantly lags behind building blocks with standard or no bottleneck.

4.7. Squeeze-and-Excitation Unit

A squeeze-and-excitation (SE) [4] block is an architectural unit for convolutional neural networks that performs dynamic channel-wise feature recalibration to improve the representation power of a convolutional building block. Given input **X**, the SE unit first squeezes the spatial information using global average pooling:

$$z_c = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} x_c(i,j), \tag{2}$$

This step is followed by an excitation step that recalibrates the channels:

$$\mathbf{s} = \sigma(W_2(\mathrm{ReLU}(W_1(\mathbf{z})))). \tag{3}$$

where W_1 and W_2 are two learned linear transformations and σ refers to the sigmoid activation function. The final output **Y** of the SE unit is obtained by scaling the original inputs $\mathbf{X}$:$\mathbf{Y} = \mathbf{X} \cdot \mathbf{s}$, where $\cdot$ refers to the channel-wise multiplication.

MnasNet [15] brought the SE unit to mobile vision networks, as they were shown to improve model performance compared to previous state-of-the-art models with similar latency. More recently, however, squeeze-and-excitation modules have been removed from mobile vision architectures when they are deployed on embedded ML accelerators. EfficientNet-lite [32] and MobileNetEdgeTPUv2 [33], for example, remove the SE unit as they claim it is not well supported for mobile accelerators, such as the Google EdgeTPU.

We evaluated the SE block by comparing design spaces with the depthwise separable building block, with and without an SE unit extension. Figure 8 shows that the SE block is a favorable addition for certain mobile platforms as it is very FLOPS-efficient. This translates to it being more optimal for mobile CPUs and GPUs. However, for the embedded Jetson Nano GPU, it is no longer a clear benefit to include the SE unit as it fails to exploit the optimizations for convolutional kernels and adds a significant delay. Comparing models with and without the SE unit and with equal execution time on a mobile CPU, we found that those same models without the SE unit have a running time that is up to a factor of 1.9× longer on the embedded Jetson Nano GPU.

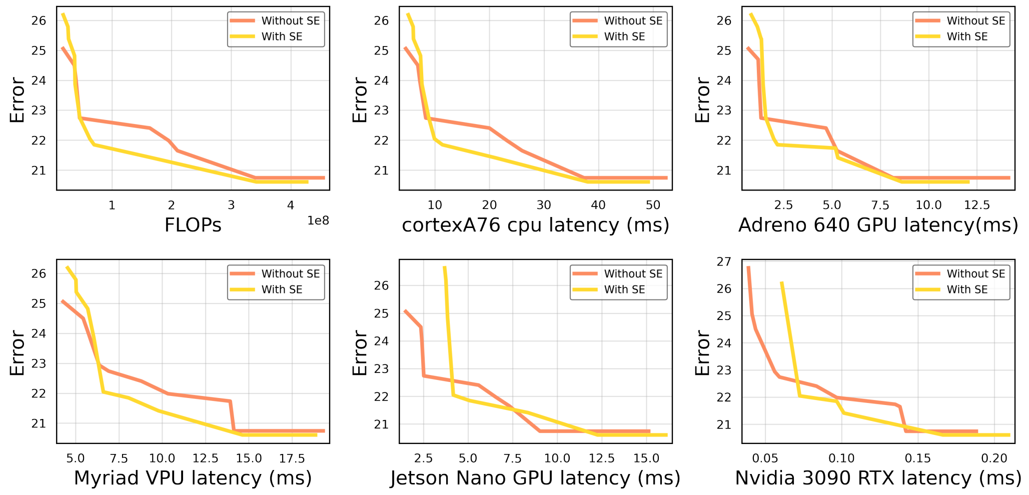


Figure 8. Adding a squeeze-and-excitation block improves performance on most hardware platforms but not all. The Jetson Nano GPU, for example, executes them significantly slower, making them not always the optimal choice.

5. Conclusions

Developing a new model for a targeted hardware platform, and with given design constraints, requires careful consideration of the building block. An evaluation based on FLOPs often leads to misleading conclusions with respect to the relative benefits of the components in the standard MBConv block on specific hardware platforms. We developed a methodology that can be used to select building blocks and constrain the design spaces as a first step in the network design process (i.e., as a preselection step before NAS) in a hardware-aware manner, while maintaining the freedom to trade-off between task performance and execution efficiency.

We used our approach to evaluate the hardware efficiency of different convolutional building blocks on various hardware platforms. Our results show that execution of building blocks with components that are theoretically efficient, such as the SE unit, take a factor of 1.9× longer to execute than their non-optimized counterparts due to better hardware utilization on platforms with specific embedded ML accelerators, such as the Intel NCS2 or the Nvidia Jetson Nano.

In essence, the insights gained from this investigation highlight the importance of benchmarking the building blocks used in mobile vision neural networks, which has been overlooked in the past. We believe that our methodology will be key to the development of hardware-aware neural networks for deployment on edge devices.

Author Contributions: Conceptualization, M.B., M.F., M.V. and J.D.; Methodology, M.B., M.F., M.V. and J.D.; Formal analysis, M.B.; Resources, M.F., M.V. and J.D.; Writing—original draft, M.B.; Writing—review & editing, M.B., M.V. and J.D.; Visualization, M.B.; Supervision, M.F., M.V. and J.D.; Funding acquisition, M.V. and J.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received funding through the Research Foundation Flanders (FWO-Vlaanderen) under grant G006718N and 1S47820N.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
2. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520.
3. Howard, A.; Sandler, M.; Chu, G.; Chen, L.C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; et al. Searching for mobilenetv3. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–28 November 2019; pp. 1314–1324.
4. Hu, J.; Shen, L.; Sun, G. Squeeze-and-excitation networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 7132–7141.
5. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 6848–6856.
6. Ma, N.; Zhang, X.; Zheng, H.T.; Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 116–131.
7. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
8. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.
9. Baker, B.; Gupta, O.; Naik, N.; Raskar, R. Designing neural network architectures using reinforcement learning. *arXiv* **2016**, arXiv:1611.02167.
10. Liu, H.; Simonyan, K.; Yang, Y. Darts: Differentiable architecture search. *arXiv* **2018**, arXiv:1806.09055.
11. Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.J.; Fei-Fei, L.; Yuille, A.; Huang, J.; Murphy, K. Progressive neural architecture search. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 19–34.
12. Real, E.; Aggarwal, A.; Huang, Y.; Le, Q.V. Regularized evolution for image classifier architecture search. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 4780–4789.
13. Pham, H.; Guan, M.; Zoph, B.; Le, Q.; Dean, J. Efficient neural architecture search via parameters sharing. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 4095–4104.
14. Stamoulis, D.; Ding, R.; Wang, D.; Lymberopoulos, D.; Priyantha, B.; Liu, J.; Marculescu, D. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Würzburg, Germany, 16–20 September 2019; pp. 481–497.
15. Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; Le, Q.V. Mnasnet: Platform-aware neural architecture search for mobile. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 2820–2828.
16. Wu, B.; Dai, X.; Zhang, P.; Wang, Y.; Sun, F.; Wu, Y.; Tian, Y.; Vajda, P.; Jia, Y.; Keutzer, K. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 10734–10742.
17. Guo, Z.; Zhang, X.; Mu, H.; Heng, W.; Liu, Z.; Wei, Y.; Sun, J. Single path one-shot neural architecture search with uniform sampling. In Proceedings of the European Conference on Computer Vision, Glasgow, UK, 23–28 August 2020; pp. 544–560.
18. Cai, H.; Gan, C.; Wang, T.; Zhang, Z.; Han, S. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv* **2019**, arXiv:1908.09791.
19. Radosavovic, I.; Johnson, J.; Xie, S.; Lo, W.Y.; Dollár, P. On network design spaces for visual recognition. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–28 November 2019; pp. 1882–1890.
20. Radosavovic, I.; Kosaraju, R.P.; Girshick, R.; He, K.; Dollár, P. Designing network design spaces. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 10428–10436.
21. Zoph, B.; Vasudevan, V.; Shlens, J.; Le, Q.V. Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 8697–8710.
22. Satopaa, V.; Albrecht, J.; Irwin, D.; Raghavan, B. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In Proceedings of the 2011 31st International Conference on Distributed Computing Systems Workshops, Minneapolis, MN, USA, 20–24 June 2011; pp. 166–171.
23. Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; He, K. Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1492–1500.
24. Gupta, S.; Akin, B. Accelerator-aware neural network design using automl. *arXiv* **2020**, arXiv:2003.02838.
25. Tan, M.; Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; pp. 6105–6114.
26. Boroumand, A.; Ghose, S.; Akin, B.; Narayanaswami, R.; Oliveira, G.F.; Ma, X.; Shiu, E.; Mutlu, O. Google neural network models for edge devices: Analyzing and mitigating machine learning inference bottlenecks. In Proceedings of the 2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT), Atlanta, GA, USA, 26–29 September 2021; pp. 159–172.

27. Chowdhery, A.; Warden, P.; Shlens, J.; Howard, A.; Rhodes, R. Visual wake words dataset. *arXiv* **2019**, arXiv:1906.05721.
28. Zhang, L.L.; Han, S.; Wei, J.; Zheng, N.; Cao, T.; Yang, Y.; Liu, Y. nn-Meter: Towards Accurate Latency Prediction of Deep-Learning Model Inference on Diverse Edge Devices. In Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services, Virtual, 24 June–2 July 2021; ACM: New York, NY, USA, 2021; pp. 81–93. [CrossRef]
29. Xiong, Y.; Liu, H.; Gupta, S.; Akin, B.; Bender, G.; Wang, Y.; Kindermans, P.J.; Tan, M.; Singh, V.; Chen, B. Mobiledets: Searching for object detection architectures for mobile accelerators. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 3825–3834.
30. Tan, M.; Le, Q. Efficientnetv2: Smaller models and faster training. In Proceedings of the International Conference on Machine Learning, Virtual, 13–14 August 2021; pp. 10096–10106.
31. Gibson, P.; Cano, J.; Turner, J.; Crowley, E.J.; O'Boyle, M.; Storkey, A. Optimizing grouped convolutions on edge devices. In Proceedings of the 2020 IEEE 31st International Conference on Application-Specific Systems, Architectures and Processors (ASAP), Manchester, UK, 6–8 July 2020; pp. 189–196.
32. Google. EfficientNet-Lite. Available online: https://github.com/tensorflow/tpu/blob/master/models/official/efficientnet/lite/README.md (accessed on 15 August 2022).
33. Akin, B.; Gupta, S.; Long, Y.; Spiridonov, A.; Wang, Z.; White, M.; Xu, H.; Zhou, P.; Zhou, Y. Searching for Efficient Neural Architectures for On-Device ML on Edge TPUs. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 19–20 June 2022; pp. 2667–2676.