BackboneAnalysis: Structured Insights into Compute Platforms from CNN Inference Latency

Hafner, Frank M.; Zeller, Matthias ; Schutera, Mark ; Abhau, Jochen ; Kooij, J.F.P.

DOI
10.1109/IV51971.2022.9827260

Publication date
2022

Document Version
Final published version

Published in
Proceedings of the 2022 IEEE Intelligent Vehicles Symposium (IV)

2022 IEEE Intelligent Vehicles Symposium (IV)
June 5-9, 2022. Aachen, Germany

2022 IEEE Intelligent Vehicles Symposium (IV) | 978-1-6654-8821-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/IV51971.2022.9827260

BackboneAnalysis: Structured Insights into Compute Platforms from CNN Inference Latency

Frank M. Hafner[1], Matthias Zeller[1], Mark Schutera[1], Jochen Abhau[1] and Julian F.P. Kooij[2]

***Abstract*— Customization of a convolutional neural network (CNN) to a specific compute platform involves finding an optimal pareto state between computational complexity of the CNN and resulting throughput in operations per second on the compute platform. However, existing inference performance benchmarks compare complete backbones that entail many differences between their CNN configurations, which do not provide insights in how fine-grade layer design choices affect this balance.**

We present BackboneAnalysis, a methodology for extracting structured insights into the trade-off for a chosen target compute platform. Within a one-factor-at-a-time analysis setup, CNN architectures are systematically varied and evaluated based on throughput and latency measurements irrespective of model accuracy. Thereby, we investigate the configuration factors input shape, batch size, kernel size and convolutional layer type.

In our experiments, we deploy BackboneAnalysis on a Xavier iGPU and a Coral Edge TPU accelerator. The analysis reveals that the general assumption from optimal Roofline performance that higher operation density in CNNs leads to higher throughput does not always hold. These results highlight the importance for a neural network architect to be aware of platform-specific latency and throughput behavior in order to derive sensible configuration decisions for a custom CNN.

I. Introduction

Algorithms based on convolutional neural networks (CNN) have demonstrated to be state-of-the-art in many major computer vision tasks such as image classification [10], object detection [12] and semantic segmentation [3], [25]. In response, a strong demand in industry arised for acceleration of CNN inference often referred to as AI accelerators [7], [16]. While GPUs are well suited by design, also new compute hardware was introduced [15].

The majority of these compute platforms have in common that they leverage single instruction multiple data (SIMD) concepts for highly parallel processing of CNNs. Roofline [31], as a renowned technique for the evaluation of the efficiency of compute workloads, was applied to AI accelerators and GPUs for CNNs [13], [16]. From the Roofline analysis for CNNs [13], [16], [31] one might conclude that *higher operational density in CNNs (in terms of operations per byte transferred from memory) increases throughput (in operations per second) on a compute platform up to a computational bound.* However, as we will show in our experiments, this is *not* generally the case and a proper analysis of configuration decisions in CNN architectures is needed on a per platform basis.

When deploying a CNN on a compute platform several steps have to be completed. In a first step, a suitable compute platform needs to be chosen. Existing CNN inference hardware benchmarks such as MLPerf [22] target this use case and support the decision with insights in the latency of state-of-the-art neural network architectures [6], [19], [21], [36]. In a second step, neural network architects face the task of customizing a target neural network configuration to a given compute platform such as in selecting convolutional block types or kernel sizes. Typically, this challenge is phrased as finding a pareto optimum between accuracy (higher is better) and latency (lower is better) such as seen in Neural Architecture Search (NAS) [14], [32].

However, to minimize latency one needs to consider how more finegrained network design choices, such as input shape, batch size, kernel size and convolutional layer type affect another complex

[1]Autonomous Mobility Systems, ZF Friedrichshafen AG, Germany firstname.lastname@zf.com

[2]Intelligent Vehicles Group, Delft University of Technology, Netherlands j.f.p.kooij@tudelft.nl

TABLE I: Selected differences of CNNs in MLPerf and the respective computational complexity and throughput on NVIDIA Xavier [22].

	SSD-MobileNet-V1	**Resnet50 v1.5**	**SSD-ResNet34**
Input Size	300x300	224x224	1200x1200
Conv. Block	Depthwise Sep. Conv.	Bottleneck Stand. Conv.	Standard Convolution
Head	SSD Head	Fully Connected Layers	SSD Head
Comp. Compl.	2.47 GOPs	8.2 GOPs	433 GOPs
Throughput	4.05 TOP/s	3.83 TOP/s	9.22 TOP/s

pareto state, namely between the computational complexity of the network (lower is better) and its throughput in operations per second for a specific compute platform (higher is better). While the existing benchmarks focus on the first stage, the proposed BackboneAnalysis supports CNN architects during this second stage to understand these trade-offs for their available compute platforms. The example of Table I illustrates that existing benchmarks, in this case MLPerf [22], do not provide insight in the finegrained design choices. BackboneAnalysis fills this gap.

The main contributions of this work are:

- BackboneAnalysis, a novel methodology systematically varying one-factor-at-a-time for supporting the search of the pareto optimum between computational complexity and throughput. The implementation proposed in this work considers the influence of the CNN configuration factors convolutional block types, kernel sizes, batch sizes and input sizes.
- We apply the methodology to the compute platforms Xavier iGPU and Coral TPU. Our results show that, to some extent contrary to expectations from optimal Roofline performance, higher operational density does often not lead to higher throughput. This underlines that a more fine-grained analysis is needed and BackboneAnalysis allows to characterize these properties of the studied platforms in detail.

II. Related Work

Although inference execution latency is reported for most new state-of-the-art CNN architectures, the metrics are measured on heterogeneous compute platforms and, therefore, incomparable. Also, for compute platforms dedicated to CNNs no standard benchmarks such as DMIPS [30] for CPUs exist. As a result, several deep learning benchmarks have been proposed recently [5], [6], [9], [19], [21], [22], [36]. According to [36] existing benchmarks are divided into training [5], [9] and inference benchmarks [6], [19], [21], [22], [36].

A common approach of benchmarking deep neural network inference is the assessment of basic mathematical operations such as matrix multiplication or max pooling [1], [6] or analyzing single layers of a CNN on compute platforms [17]. It was shown that the theoretical number of achievable operations per second (OP/s) or throughput often reported by compute hardware manufacturers is not reflecting performance for CNN workloads, as this metric neglects memory-bounds and compute utilization [15], [17], [31].

Besides that, several inference benchmarks consider the execution latency of neural networks as a whole. [19], [21] and [6] analyze deep learning frameworks on compute platforms, where [6] and [21] focus on edge devices. All mentioned benchmarks have in common that they are based on state-of-the-art academic neural network architectures, such as MobileNets [12] and ResNets [10], where some include Roofline analyses [31]. Through their design, the benchmarks support the selection of a compute platforms for CNN deployment and, unlike BackboneAnalysis, do not support a neural network architect in finegrained CNN configuration decisions.

Finegrained configuration design choices are key to finetune performance, and are therefore at the core of Neural Architecture Search (NAS). Within NAS, an algorithm searches an approximately ideal neural network configuration for a task in a defined search space. While state-of-the-art results in

terms of accuracy-latency trade-off where achieved, it is shown that the consideration of hardware restrictions is critical for success [32], [37]. Recently, NAS algorithms shift the focus to actually measured execution latency in milliseconds or latency estimates based on look-up tables for compute hardware [14], [32], while still ignoring the pareto state between throughput and computational complexity. Networks from NAS often result in complex compositions of convolutional layers and interconnections and, hence, are not applicable for a neural network expert seeking an understanding of the effects of neural network configurations on execution latency or throughput on a compute platform. Additionally, the time and computational requirements of NAS grow exponentially with configuration possibilities, which limits the practicability in many applications. Therefore, NAS methods often define a search space based on heuristics or unstructured findings [34]. BackboneAnalysis takes a set of relevant configuration decisions as in NAS and implements a structured comparison on top.

III. Backbone Analysis

BackboneAnalysis is a methodology for extracting insights into compute platforms from CNN latency following a one-factor-at-a-time experimentation setup. It consists of a collection Θ of N neural network configurations θ, where $\theta \in \Theta$. In the design phase of BackboneAnalysis the core configuration factors F are defined and, subsequently, the measurements are conducted with varying one of those factor per experiment keeping the rest of the design constant.

The focus of BackboneAnalysis is on supporting neural network architects in finding a good pareto state between computational complexity of a CNN and throughput in terms of operations per second on a platform and, therefore, only latency l is measured while accuracy is neglected. Due to rapidly evolving factors of interest in CNNs we consider the implementation defined in this work (Section III-A) as an example which can be adapted by future users of the methodology.

A. Finegrained Network Configuration Choices

As the baseline structure for all neural network configurations Θ for this study, we propose ResNet-50 [10]. The setup of ResNets in terms of width, height and channel depth in the feature maps leads to the fact that all layers in the network have the same amount of operations. This property is desirable for the analysis to obtain a sensible average of the performance of layers at different depth in the network.[1]

The key contribution of the experimentation in this work is the factor combination F for the neural network configurations Θ used for BackboneAnalysis. It is composed of convolutional block type c, kernel size k, batch size b and input shape i as those factors are typically varied between academic network configurations and relevant for execution latency. Therefore, each network configuration θ is built upon a permutation of these factors with ResNet-50 as the baseline structure. A visualization of the resulting networks is found in Table III.

A review of literature for deep learning architectures points out that a substantial share of convolutional block types c in state-of-the-art neural network architectures can be condensed to three variants which are mutually substitutable (see overview in Table II): Standard convolution block [10], [18], [26], spatially separable convolution block [27], [28] and depthwise separable convolution block [12][2]. Additionally, the described types are frequently embedded between bottleneck layers [10], [11], [24], [27], [29]. Hence, non-bottleneck as well as bottleneck convolutional block types are added as variations of c. The network configurations with different convolutional block types, ranked from high to low computational complexity, are standard convolution, spatially separable convolution and depthwise separable convolution. The same order from high to low is expected for operational density

[1] As the focus of BackboneAnalysis is performance in convolutional layers fully connected layers are removed and "straightened" convolutions are considered as in [35]. This means that all layers in a convolutional block have the same amount of channels.

[2] Grouped convolutions are a itermediary between depthwise separable and standard convolutions and, hence, neglected at this point.

TABLE II: State-of-the-art neural network architectures include convolutional block types identified for BackboneAnalysis (not comprehensive). Convolutional block structures described with depth d, kernel size k and channel C

Convolutional block type	State-of-the-art reference architectures	Convolutional block structure
Standard convolution	VGG-16 [26], Alexnet [18], ResNet-18, ResNet-34 [10], WRN [35]	$k \times k \times d$
Bottleneck standard convolution	ResNet-50, ResNet-101, ResNet-152 [10], ResNeXt [33], DPN [4]	$1 \times 1 \times d \rightarrow k \times k \times d \rightarrow 1 \times 1 \times d$
Spatially separable convolution	Inception-V2 [28], Inception-V4 [27], ERFNet [23]	$k \times 1 \times d \rightarrow 1 \times k \times d$
Bottleneck spatially separable convolution	Inception-ResNet-v1 and v2 [27]	$1 \times 1 \times d \rightarrow k \times 1 \times d \rightarrow 1 \times k \times d \rightarrow 1 \times 1 \times d$
Depthwise separable convolution	MobilenetV1 [12]	$k \times k \times 1$ for each $C \rightarrow 1 \times 1 \times d$
Bottleneck depthwise separable convolution	MobilenetV2 [24], MobilenetV3 [11], EfficientNet [29]	$1 \times 1 \times d \rightarrow k \times k \times 1$ for each $C \rightarrow 1 \times 1 \times d$

on a compute platform.

The convolutional kernel size k has a significant influence on mathematical complexity O_θ and, hence, is another key factor considered for neural network configurations. As in literature mostly kernel sizes of 3, 5 and 7 are used, the convolutional block types are varied with these kernel sizes [10]–[12], [18], [24], [26]–[29]. In general, bigger kernel sizes in network configurations are expected to lead to higher operational density and higher computational complexity.

The batch size b describes the number of inputs samples which are processed in parallel by a neural network. For the analysis batch sizes b of 1, 2, and 4 are chosen as those numbers are applicable in real-world applications [22].

State-of-the-art work shows that higher resolution input shapes i lead to higher accuracies for different tasks of convolutional neural networks [29]. In this analysis an input shape i of $224 \times 224 \times 3$ is taken as the baseline from [10]. Additionally, a scaling of input width and height by 2.5 leads to input shapes of $560 \times 560 \times 3$ (similar to VGA resolution) and $1400 \times 1400 \times 3$ (similar to Full HD resolution).

Batch size and input shape linearly scale the computational complexity of the network configurations. Additionally, with increasing batch sizes and input shapes the operational density increases.

Overall, the benchmark varies four factors with influence on execution latency in a one-factor-at-a-time setup (see Table III). Details for each network such as computational complexity and number of parameters as well as latency are found in a reference CSV sheet [3].

B. Metrics

We propose to measure the execution latency $l_{\theta,P}$ of each of the network configurations $\theta \in \Theta$ on a selected compute platform P consisting of the compute hardware and its overlaying software stacks. Hereby, execution latency in milliseconds is defined as

$$l_{\theta,P} = t_1 - t_0 \qquad [ms] \quad (1)$$

where t_0 is the timestamp when the input is available for the first layer of θ and t_1 is the timestamp when the output of the last layer of θ is available for further processing on P. This definition implies that hardware latency is considered for BackboneAnalysis and system latency including data pipelining and data transfer is neglected [8].

The computational complexity of a neural network configuration θ is defined by the number of mathematical operations O_θ for inference. Given the computational complexity O_θ and the execution latency $l_{\theta,P}$, the operations per second (OP/s) carried out on P can be calculated as

$$O_{\theta,P} = \frac{O_\theta}{l_{\theta,P}} \times 1000 \qquad [\tfrac{OP}{s}] \quad (2)$$

which is referred to as the throughput on P for a network configuration θ [2], [20]. Due to possible quantization the commonly used metric floating-point operations per second (FLOP/s) is generalized to operations per second (OP/s) in this work similar to [17].

[3]Details for all models (open with Chrome): `https://figshare.com/s/682b765e575abdf7a0fa`

TABLE III: Network configurations for Backbone Benchmark. Last non-bottleneck convolution layer in each block has a stride of 2 to downscale the feature maps. Batch normalization and ReLU activation follow on each convolutional layer. Residual connections according to ResNet-50 [10] and illustration inspired by [10].

Input shape i	$\{224 \times 224 \times 3, 560 \times 560 \times 3, 1400 \times 1400 \times 3\}$
Batch size b	$\{1, 2, 4\}$
Kernel sizes k	$\{3, 5, 7\}$

name	output size	Standard conv.	Bottleneck standard conv.	Spatially separable conv.	Bottleneck spatially sep. conv.	Depthwise separable conv.	Bottleneck depthwise sep. conv.
conv1	$i/2$	$7 \times 7, 64$ stride=2					
conv2	i/4	3×3, max pool, stride=2					
		$\begin{bmatrix} k \times k, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ k \times k, 64 \\ 1 \times 1, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times k, 64 \\ k \times 1, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 1 \times k, 64 \\ k \times 1, 64 \\ 1 \times 1, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} k \times k, 1 \\ 1 \times 1, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ k \times k, 1 \\ 1 \times 1, 64 \end{bmatrix} \times 3$
conv3	$i/8$	$\begin{bmatrix} k \times k, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ k \times k, 128 \\ 1 \times 1, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times k, 128 \\ k \times 1, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 1 \times k, 128 \\ k \times 1, 128 \\ 1 \times 1, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} k \times k, 1 \\ 1 \times 1, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ k \times k, 1 \\ 1 \times 1, 128 \end{bmatrix} \times 4$
conv4	$i/16$	$\begin{bmatrix} k \times k, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ k \times k, 256 \\ 1 \times 1, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times k, 256 \\ k \times 1, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 1 \times k, 256 \\ k \times 1, 256 \\ 1 \times 1, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} k \times k, 1 \\ 1 \times 1, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ k \times k, 1 \\ 1 \times 1, 256 \end{bmatrix} \times 6$
conv5	$i/32$	$\begin{bmatrix} k \times k, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ k \times k, 512 \\ 1 \times 1, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times k, 512 \\ k \times 1, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 1 \times k, 512 \\ k \times 1, 512 \\ 1 \times 1, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} k \times k, 1 \\ 1 \times 1, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ k \times k, 1 \\ 1 \times 1, 512 \end{bmatrix} \times 3$

Another metric to describe a neural network configuration θ is operational density ρ_θ. operational density is defined as the average number of operations per unit of information transferred from memory (such as a 8-bit units for int8 quantized models) [17]. Hence, operational density for neural networks can be estimated as follows:

$$\rho_\theta = \frac{O_\theta}{b} \qquad [\tfrac{OP}{byte}] \quad (3)$$

where b refers to the number of bytes transferred from memory (mostly feature maps and kernel weights). In generall, operational density can differ between compute platforms due to different memory and cache layouts.

Overall the above defined concepts of execution latency $l_{\theta,P}$, computational complexity O_θ, throughput $O_{\theta,P}$ and operational density ρ_θ cumulate to a diverse set of metrics to discuss compute platforms by means of CNN configurations, while only execution latency $l_{\theta,P}$ needs to be physically measured to execute the analysis. In this work, the average latency and the respective standard deviation of 100 executions is logged.

IV. Experiments

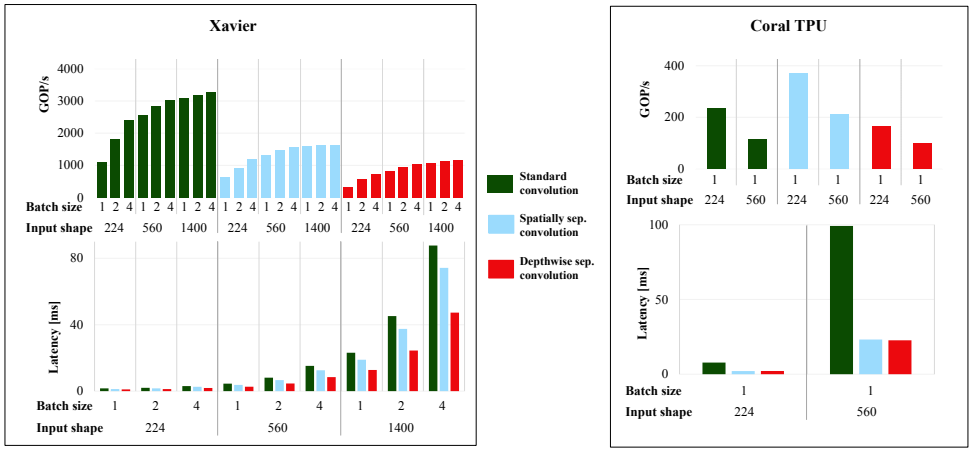
After a discussion of the implementation in section IV-A, we will evaluate the influence of the factors input size, batch size, convolutional layer type and kernel size on a Xavier iGPU and a Coral TPU based on BackboneAnalysis in section IV-B.

A. Implementation

Table IV lists the compute platforms with hardware and software stacks which are examined in this section. For assessment of the Xavier iGPU the 30 Watt mode (MODE_30W_ALL) is chosen and trtexec is used for the execution latency measurement. As the non-deterministic optimization of TensorRT occasionally leads to a slightly differing performances the lowest average latencies of three optimization runs is logged for configurations with the two smaller input shapes. For Coral TPU libedgetpu1-max is installed. For both compute platforms int8 quantization is used. With Xavier a full execution of BackboneBenchmark takes roughly 4 days, with Coral TPU around 1.5 days.

TABLE IV: Compute platforms examined in section IV.

Abbreviation	Device name	Compute platform	
		Compute hardware	Software stack
Xavier iGPU	NVIDIA Jetson AGX Xavier Developer Kit	iGPU, 32GB LPDDR4x	Models from ONNX Jetpack 4.3 (TensorRT 6.0.1, cuDNN 7.6.3, CUDA 10.0.326)
Coral TPU	Coral USB Accelerator with Lenovo IdeaPad 5i 14	Edge TPU, USB 3.0 Type-C Intel Core i5-1035G1 8 GB DDR4	Models from Tensorflow Lite Edge TPU Compiler 15.0.340273435 Tensorflow Lite 2.5.0, Ubuntu 16.04



(a) Xavier iGPU. (b) Coral TPU.

Fig. 1: Analysis of input shapes and batch sizes for configurations with different convolutional block types on Xavier iGPU and Coral TPU ($k = 3$). Throughput in GOP/s. Higher is better. Execution latency in ms. Lower is better.

B. Assessment of Xavier iGPU and Coral TPU

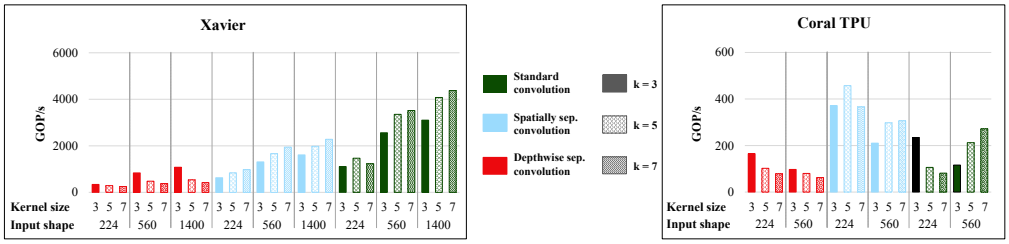
In the following the influence of the four emphasized factors input size, batch size, kernel size and convolutional layer type will be analyzed on the compute platforms Xavier iGPU and Coral TPU (for measured latencies of all models see `https://figshare.com/s/682b765e575abdf7a0fa`).

1) Model Compilation: Before analyzing CNN latency behavior BackboneAnalysis reveals which network configurations can be compiled for a compute platform.

For Xavier all configurations were compiled, while for Coral TPU none of the models with input shape 1400 was successfully compiled. Also, Coral TPU does not support batch sizes bigger than 1. This hints that the compute platform Coral TPU was designed for workloads with lower computational complexity, while Xavier iGPU is flexible in this regard.

2) Batch Size, Input Shape and Convolutional Layer Type: Larger batch sizes and input shapes lead to higher operational density and, therefore, according to optimal Roofline performance an increase in throughput is expected. The effects seen on iGPU and TPU are visualized in Figure 1.

For Xavier, the influence of batch size and input shape are as expected (see Figure 1a, top). The throughput scales with higher operational density and a convergence is observed for all three convolutional block types. Remarkably, the reduction



(a) Xavier iGPU.

(b) Coral TPU.

Fig. 2: Throughput of configurations with varying kernel sizes on Xavier iGPU and Coral TPU ($b = 1$). Higher is better.

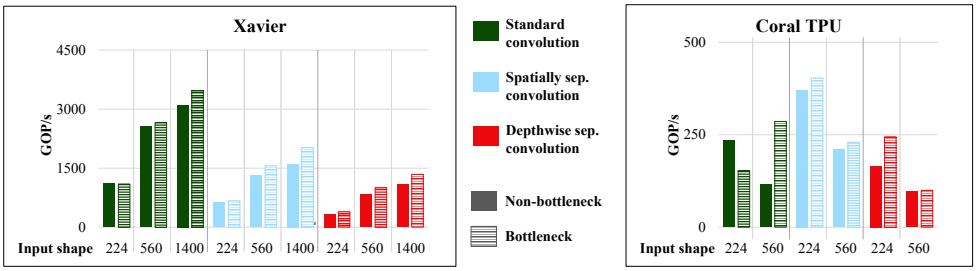
of computational complexity with different convolutional block types is not reflected in a similarly lowered latency on Xavier (see Figure 1a, bottom). While configurations with spatially separable convolutions have an average reduction of 57.5% in computational complexity in comparison to configurations with standard convolutions, only an average reduction of 17.6% in latency is seen. Therefore, a good tradeoff between throughput and computational complexity is observed for standard convolutions. The same fact holds for configurations with depthwise separable convolutions with a reduction in computational complexity of 81.7% but an average reduction of latency of only 41.2%. Hence, the analysis shows that simply reducing computational complexity in configurations for Xavier iGPU with alternative convolutional blocks can be punished with disproportionately lower throughput.

The measurements on Coral TPU also reveal interesting performance behavior (see Figure 1b). For all configurations a decrease in throughput is seen when a larger input shape (i.e. larger operational density) is used. This contradicts the general assumptions from optimal Roofline models. Even more remarkable is that configurations with spatially separable convolutions achieve a higher throughput than standard convolutions, and configurations with depthwise convolutions are almost on a par with standard convolutions in this metric. It follows that contrary to the optimal Roofline assumptions, the Coral TPU is performing low in throughput on configurations with standard convolutions. With alternative convolutional block types a reduction in average latency of around 75% is observed (see Figure 1b, bottom).

3) Kernel Size: The throughput for Xavier iGPU and Coral TPU for different kernel sizes are shown in Figure 2. For Xavier iGPU an increase in throughput is seen for bigger kernel sizes for configurations with standard as well as spatially separable convolutions (see Figure 2a). Again this is explainable by higher operational density in these configurations. Unexpectedly, for configurations with depthwise separable convolutions a kernel size of 3 is most performant. Also for Coral TPU, all configurations with the smaller input shape show higher throughput with a kernel size of 3 (see Figure 2b).

4) Bottleneck Blocks: In Figure 3 a comparison of latency and throughput behavior of Xavier iGPU and Coral TPU for configurations with bottleneck and non-bottleneck blocks is visualized.

The throughput for all configurations on Xavier iGPU is increased when bottleneck layers are introduced (on average by 14%, see Figure 3a). Again, this is unexpected based on optimal Roofline models and suggests that the usage of bottleneck layers is advisable for Xavier. For Coral TPU non-standard convolutional blocks increase the throughput with bottleneck layers similarly to the Xavier iGPU. Hence, for both compute platforms adding



(a) Xavier iGPU.

(b) Coral TPU.

Fig. 3: Analysis of bottleneck layers on Xavier iGPU and Coral TPU ($k = 3, b = 1$). Throughput in GOP/s. Higher is better.

bottleneck layers is promising as a good tradeoff between computational complexity and throughput is observed.

In summary, the discussion of experiments above shows how BackboneBenchmark can provide insights in the tradeoff between computational complexity and throughput based on configuration decisions.

V. Conclusion

In this work we presented BackboneAnalysis as a methodology for extracting insights into compute platforms for neural network architects in order to find better pareto states between computational complexity of neural networks and throughput on a compute platform. In the application of the method in this paper we analyzed the factors convolutional block types, input shapes, batch sizes and kernel sizes on a Xavier iGPU and a Coral Edge TPU accelerator.

The analysis shows that for Xavier iGPUs a high operational density in network architectures, such as with large input shapes and batch sizes, is crucial for high throughput. Therefore, also replacing standard convolution with less compute intense alternatives is not always beneficial as those convolutional block types lead to significantly lower throughput. In contrast, the Coral EdgeTPU benefits from computationally cheaper convolution types and is performing low with standard convolutions. On top of that, both platforms can benefit in terms of throughput from additional bottleneck layers in many cases. While higher batch sizes and input sizes are beneficial for the throughput in a Xavier iGPU, CoralTPU loses performance with bigger input shapes and does not support batching.

Overall, the described insights support a CNN architect to understand the tradeoff between computational complexity and throughput on a compute platform when customizing a neural network. In the future, we expect that BackboneAnalysis could be succesfully applied as a pre-processing step to find a reasonable search space for network architecture search (NAS) methods in the future.

ACKNOWLEDGMENT

This work is in part funded by the German Federal Ministry for Economic Affairs and Energy (BMWi) through the grant 19A19013Q, project ”KI Delta Learning”.

References

[1] Baidu. Deepbench: Benchmarking deep learning operations on different hardware. `https://github.com/baidu-research/DeepBench`, 2017.

[2] Cavigelli, L., Magno, M., and Benini, L. Accelerating real-time embedded scene labeling with convolutional networks. In *DAC* (2015), IEEE, pp. 1–6.

[3] Chen, L.-C., Papandreou, G., Schroff, F., and Adam, H. Rethinking atrous convolution for semantic image segmentation. *arXiv:1706.05587* (2017).

[4] CHEN, Y., LI, J., XIAO, H., JIN, X., YAN, S., AND FENG, J. Dual path networks. In *NeurIPS* (2017), pp. 4467–4475.

[5] COLEMAN, C., NARAYANAN, D., KANG, D., ZHAO, T., ZHANG, J., NARDI, L., BAILIS, P., OLUKOTUN, K., RÉ, C., AND ZAHARIA, M. Dawnbench: An end-to-end deep learning benchmark and competition. *NeurIPS ML Systems Workshop* (2017).

[6] DEUSCHLE, A. V., AND MARKL, V. End-to-end benchmarking of deep learning platforms. In *TPCTC Workshop* (2019).

[7] FOWERS, J., OVTCHAROV, K., PAPAMICHAEL, M., MASSENGILL, T., LIU, M., LO, D., ALKALAY, S., HASELMAN, M., ADAMS, L., GHANDI, M., ET AL. A configurable cloud-scale dnn processor for real-time ai. In *ISCA* (2018), IEEE, pp. 1–14.

[8] GAO, C., BRAUN, S., KISELEV, I., ANUMULA, J., DELBRUCK, T., AND LIU, S.-C. Real-time speech recognition for iot purpose using a delta recurrent neural network accelerator. In *ISCAS* (2019), IEEE, pp. 1–5.

[9] GAO, W., ZHAN, J., WANG, L., LUO, C., ET AL. Bigdatabench: A dwarf-based big data and ai benchmark suite. *arXiv:1802.08254* (2018).

[10] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *CVPR* (2016), pp. 770–778.

[11] HOWARD, A., SANDLER, M., CHU, G., CHEN, L.-C., ET AL. Searching for mobilenetv3. In *ICCV* (2019), pp. 1314–1324.

[12] HOWARD, A. G., ZHU, M., CHEN, B., KALENICHENKO, D., WANG, W., WEYAND, T., ANDREETTO, M., AND ADAM, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861* (2017).

[13] JAVED, M. H., IBRAHIM, K. Z., AND LU, X. Performance analysis of deep learning workloads using roofline trajectories. *CCF Transactions on High Performance Computing 1*, 3 (2019), 224–239.

[14] JIANG, W., YANG, L., SHA, E. H.-M., ZHUGE, Q., GU, S., DASGUPTA, S., SHI, Y., AND HU, J. Hardware/software co-exploration of neural architectures. *TCAD* (2020).

[15] JIANG, Z., LI, J., AND ZHAN, J. The pitfall of evaluating performance on emerging ai accelerators. *arXiv:1911.02987* (2019).

[16] JOUPPI, N. P., YOUNG, C., PATIL, N., PATTERSON, D., AGRAWAL, G., BAJWA, R., BATES, S., BHATIA, S., BODEN, N., BORCHERS, A., ET AL. In-datacenter performance analysis of a tensor processing unit. In *ISCA* (2017), ACM/IEEE, pp. 1–12.

[17] KARBACHEVSKY, A., BASKIN, C., ZHELTONOZSHKII, E., YERMOLIN, Y., GABBAY, F., BRONSTEIN, A. M., AND MENDELSON, A. Hcm: Hardware-aware complexity metric for neural network architectures. *arXiv:2004.08906* (2020).

[18] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. In *NeurIPS* (2012), pp. 1097–1105.

[19] LIU, L., WU, Y., WEI, W., CAO, W., SAHIN, S., AND ZHANG, Q. Benchmarking deep learning frameworks: Design considerations, metrics and beyond. In *ICDCS* (2018), IEEE, pp. 1258–1269.

[20] LIU, Z., LUO, S., XU, X., SHI, Y., AND ZHUO, C. A multi-level-optimization framework for fpga-based cellular neural network implementation. *JETC 14*, 4 (2018), 1–17.

[21] LUO, C., ZHANG, F., HUANG, C., XIONG, X., CHEN, J., WANG, L., GAO, W., YE, H., WU, T., ZHOU, R., ET AL. Aiot bench: Towards comprehensive benchmarking mobile and embedded device intelligence. In *Int. Symposium Bench* (2018), Springer, pp. 31–35.

[22] REDDI, V. J., CHENG, C., KANTER, D., MATTSON, P., SCHMUELLING, G., WU, C.-J., ANDERSON, B., BREUGHE, M., CHARLEBOIS, M., CHOU, W., ET AL. Mlperf inference benchmark. *arXiv:1911.02549* (2019).

[23] ROMERA, E., ALVAREZ, J. M., BERGASA, L. M., AND ARROYO, R. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *ITS 19*, 1 (2017), 263–272.

[24] SANDLER, M., HOWARD, A., ZHU, M., ZHMOGINOV, A., AND CHEN, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR* (2018), IEEE, pp. 4510–4520.

[25] SCHUTERA, M., JUST, S., GIERTEN, J., MIKUT, R., REISCHL, M., AND PYLATIUK, C. Machine learning methods for automated quantification of ventricular dimensions. *Zebrafish 16*, 6 (2019), 542–545.

[26] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556* (2014).

[27] SZEGEDY, C., IOFFE, S., VANHOUCKE, V., AND ALEMI, A. A. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI* (2017).

[28] SZEGEDY, C., VANHOUCKE, V., IOFFE, S., SHLENS, J., AND WOJNA, Z. Rethinking the inception architecture for computer vision. In *CVPR* (2016), IEEE, pp. 2818–2826.

[29] TAN, M., AND LE, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv:1905.11946* (2019).

[30] WEICKER, R. P. Dhrystone: a synthetic systems programming benchmark. *Communications of the ACM 27*, 10 (1984), 1013–1030.

[31] WILLIAMS, S., WATERMAN, A., AND PATTERSON, D. Roofline: an insightful visual performance model for multicore architectures. *Comm. of the ACM 52*, 4 (2009), 65–76.

[32] WU, B., DAI, X., ZHANG, P., WANG, Y., ET AL. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR* (2019), pp. 10734–10742.

[33] XIE, S., GIRSHICK, R., DOLLÁR, P., TU, Z., AND HE, K. Aggregated residual transformations for deep neural networks. In *CVPR* (2017), pp. 1492–1500.

[34] XIONG, Y., LIU, H., GUPTA, S., AKIN, B., ET AL. Mobiledets: Searching for object detection architectures for mobile accelerators. *arXiv:2004.14525* (2020).

[35] ZAGORUYKO, S., AND KOMODAKIS, N. Wide residual networks. *arXiv:1605.07146* (2016).

[36] ZHANG, Q., ZHA, L., LIN, J., TU, D., LI, M., LIANG, F., WU, R., AND LU, X. A survey on deep learning benchmarks: Do we still need new ones? In *Int. Symposium Bench* (2018), Springer, pp. 36–49.

[37] ZOPH, B., AND LE, Q. V. Neural architecture search with reinforcement learning. *arXiv:1611.01578* (2016).