Bridging the MAC Tunnel Vision: System-Level Performance Analysis of CNN Model Deployment at the Edge

Dwith Chenna

AMD http://www.amd.com
dwith.chenna@ieee.org

***Abstract*— Convolutional Neural Networks (CNNs), widely used in computer vision tasks, require substantial computation and memory resources, making it challenging for these models to run efficiently on resource-constrained devices. Network Architecture Search (NAS) methods have been developed to design compute-efficient models like MobileNet and EfficientNet. However, many of these models suffer from inefficiencies in hardware utilization due to their ineffectiveness in understanding the system-level details like software framework, memory bandwidth limitations and hardware capabilities for model deployment on devices. This excessive focus on compute efficiency, sometimes referred to as the "MAC tunnel vision" problem, leads to sub-optimal performance during model deployment. In this paper, we aim to bridge this gap by analyzing the performance across popular network architectures like ResNet, EfficientNet for different network hyper-parameters such as input size, feature dimensions, grouped convolution and network depth. This analysis provides CNN modeling engineers with the necessary tools to design models that can efficiently utilize the resources of available hardware. This approach not only incorporates hardware awareness into the model deployment but also considers different aspects of deployment, such as optimization algorithms (e.g., layer fusion, quantization), execution model, efficient kernels and hardware capabilities.**

***Index Terms*— Convolutional Neural Networks, Computer Vision, Optimization Algorithms**

I. INTRODUCTION

Convolutional Neural Networks (CNNs) have significantly impacted Embedded Vision and Edge AI by enabling AI applications on resource-constrained devices. With compute requirements of AI models growing each year, hardware accelerators have become crucial for efficiency. The ubiquitous presence of CNN models for vision applications has spurred the development of various CNN accelerator platforms [1-3]. These accelerators are designed to handle a wide range of CNN models and deliver efficient performance. However, the fast-evolving nature of CNN architectures presents a persistent challenge. This results in hardware accelerators constantly striving to keep pace with the latest models. Given the longer refresh cycles and lifecycles of hardware, this issue is increasingly prevalent across various fields. Tradeoffs related to compute and memory bandwidth may need to be reconsidered for future hardware versions. Bridging the gap between these developed models and inference platforms is necessary for overall system efficiency and enabling Edge AI applications. Achieving optimal system performance requires a deep understanding of hardware capabilities and the selection or design of architectures better suited to the hardware. CNN model deployments on hardware involves optimizing algorithms (i.e. layer fusion, pruning, quantization), execution models, efficient kernels, and leveraging hardware capabilities. Network Architecture Search (NAS) methods [4] for automatic search and design of CNN models have been used to find the optimum tradeoff between accuracy and compute efficiency. This led to development of many smaller and compute efficient model architectures like MobileNet[5] and EfficientNet[6]. A significant limitation of current NAS designs is a lack of consideration for hardware platform capabilities or features, focusing solely on compute or Multiply and Accumulate (MAC) efficiency. This results in inefficient design choices, also referred to as "MAC tunnel vision," where memory bandwidth constraints and other system and hardware limitations are overlooked.

While the goal of NAS methods is to reduce compute or the number of parameters, this does not always lead to improved inference speed, as the software framework and underlying hardware capabilities play a crucial role. For instance, MobileNetV2[7], with 307M MACs and a model size of 13MB, exhibits a 30% higher runtime compared to ResNet18[8], which has 1.8B MACs and a model size of 45MB. This comparison clearly demonstrates that the total MACs or model size does not necessarily correlate with runtime performance. Table 1 shows the performance comparison between ResNet18 and MobileNetV2 on hardware accelerator [9], highlights inefficiencies in model deployment. This discrepancy raises an important question: *how can we design models for better efficiency during deployment*? The answer involves considering various factors such as network architecture, optimizations, implementation details, and hardware capabilities. Understanding the performance characteristics of these models on the inference framework and device is crucial. Evaluating models for deployment efficiency requires a system-level approach that goes beyond just the compute or parameter count. It involves a detailed analysis of how different network design choices effects the perform under specific framework and hardware constraints. This approach ensures that the

models not only leverage the full potential of the hardware but also achieve the desired efficiency in real-world applications.

TABLE I: PERFORMANCE COMPARISON RESNET18 VS MOBILENETV2

Network	Model Size (Float)	Model Size (Quantized)	# Layers	MACs	Latency (Ms)	Throughput (fps)
ResNet18	45MB	12MB	169	1.8B	7.35	538.7fps
MobileNetV2	13MB	3.5MB	357	307M	10.34	381.95fps

Many model developers often lack knowledge or understanding of the deployment framework or device, making the deployment process even more challenging. A paradigm shift in model design that considers hardware capabilities can unlock significant performance improvements [10-11]. However, understanding the implementation details and tradeoffs related to inference hardware can be daunting. To address this, a simpler interface is needed to evaluate on-device performance, enabling its integration with automation frameworks like NAS for efficient model design. While existing benchmarks provide some guidance on the performance of different model architectures, they often fail to offer useful insights due to their limited design exploration space. Fig. 1 shows the comparison of latency (runtime) vs MACs performance of popular CNN architectures like ResNet, EfficientNet on hardware accelerator, which shows a clear gap in performance during model deployment. These results are execution runtime (ms) of CNNs on hardware accelerator using software frameworks with quantization tool.

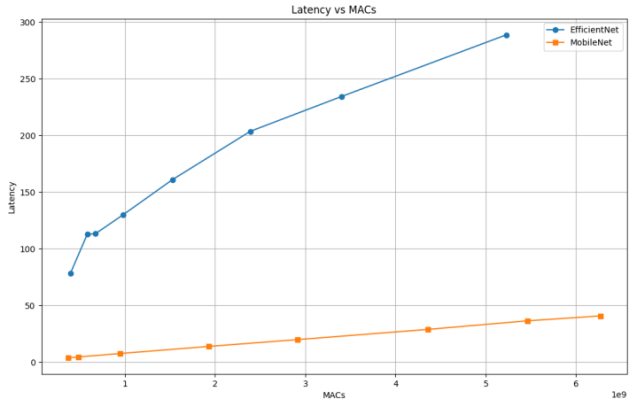

Fig. 1. Comparison of latency (ms) vs MACs performance for popular ResNet and EfficientNet architecture

In this paper, we aim to understand the effects of various macro network design parameters—such as input sizes, feature sizes, grouped convolutions and network depth—to facilitate better model design. We explore the design space, providing valuable insights for optimizing model deployment on hardware. In the following section, we will explore NAS methods and derive insights into architectural choices by examining popular network architectures. The Implementation section provides an overview of the software framework, optimizations, and hardware capabilities. Lastly, the Results and Analysis section discusses the outcomes and insights gained from performance measurements.

II. NETWORK ARCHITECTURE DESIGN

Network Architecture Search (NAS) is a powerful and increasingly essential tool in the field of machine learning and artificial intelligence, particularly for designing efficient and high-performance neural network models [4-6]. Traditional methods of manually crafting neural network architectures have become insufficient due to the rapidly growing complexity and diversity of applications in computer vision, natural language processing, and other domains. NAS automates the design process by leveraging search algorithms to explore a vast space of possible network architectures, optimizing for various performance metrics such as accuracy, model size and compute efficiency [4]. The need for NAS arises from the observation that different neural network architectures can exhibit significantly varied performance depending on the architecture, optimizations, software framework and underlying hardware. This variability poses a challenge for model developers, who must balance tradeoffs between computational cost, memory usage, and inference speed. NAS addresses this challenge by systematically evaluating a wide range of architectures, identifying optimal designs that might not be apparent through manual tuning.

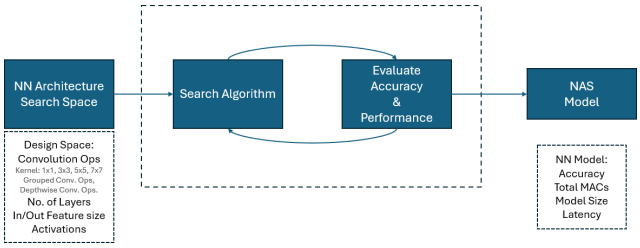

Fig. 2 Show the process of NAS design space exploration for efficient model design

Recent advancements in NAS have introduced sophisticated techniques, including reinforcement learning [12], evolutionary algorithms [13], and gradient-based methods [14], to efficiently navigate the search space. These techniques have enabled the discovery of novel architectures that outperform human-designed models on several benchmarks. Moreover, the integration of hardware- aware NAS [15] has further enhanced the applicability of these models by ensuring they are tailored to the constraints and capabilities of specific hardware platforms, such as GPUs, TPUs, and NPUs. However, in the ever-growing space of accelerator hardware there is no one solution fits all solution, which means the design choices need to be understood specific to the software framework and underlying hardware to achieve optimal performance.

In this section, we explore the different macro network

design choices and try to understand its impact on the model. We will start by examining popular network architectures like ResNet and EfficientNet to understand the tradeoffs made by NAS and its effectiveness. Next, we look at macro hyper-parameters for network architecture design like input size, in/out feature size, grouped convolution, width and network depth.

A. ResNet

ResNet, created by He et al. [8], marked a significant advancement in CNN architecture by introducing residual learning and techniques for efficient deep network training. This development addressed the vanishing gradient problem, allowing the creation of even deeper CNN models. ResNet's breakthrough enabled a 152-layer deep CNN, which won the 2015 ILSVRC competition. Compared to AlexNet and VGG, ResNet achieved 20x and 8x greater depth, respectively, with relatively lower computational complexity. Empirical evidence indicated that ResNet models with 50, 101, and 152 layers outperformed their shallower counterparts. These models demonstrated notable accuracy improvements in complex visual tasks such as image recognition and localization on the COCO dataset. ResNeXt [16] further improved upon this by considering it as an ensemble of smaller networks, employing diverse convolutions (1x1, 3x3, 5x5) alongside 1x1 bottleneck convolution blocks to explore various topologies across different paths.

B. EfficientNet

EfficientNet, developed by AutoML NAS [17], is crafted to enhance both accuracy and computational efficiency. Utilizing mobile inverted residual bottleneck convolutions (MBConv) similar to MobileNet, it adopts compound scaling [6] to create various networks tailored to different computational budgets and model sizes. EfficientNet achieved superior accuracy for compute to existing CNNs, significantly reducing model size and MACs/FLOPs. For example, EfficientNet-B0 outperforms ResNet-50 while using 5x fewer parameters and 10x fewer FLOPs. These models surpass alternatives like ResNet, DenseNet, and Inception with considerably fewer parameters.

One of the significant design choices when comparing ResNet and EfficientNet is the use of Depthwise convolution, that reduce the number of compute/MACs for the same parameter size. Fig. 3 below show the basic building blocks used to create the CNN models. The 1x1 convolutions are used to expand/compress the feature maps and 3x3 convolution allowing to work on larger feature map size within the block. Using Depthwise convolution, allowed EfficientNet to be much deeper allowing them to have a much larger receptive field and better learning capability for the same compute budget.

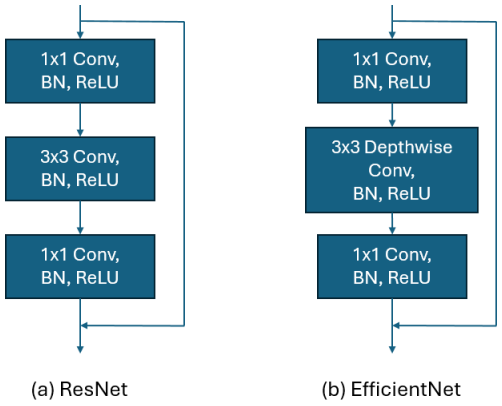

Fig. 3 Building blocks of the Convolutional Neural Network (CNN) models

The prevalence of NAS methods for model design has made these building blocks ubiquitous in many popular architectures. The design choices for such models include macro parameters that significantly influence the performance and efficiency of models. In this analysis, we start with the ResNet based NAS based on the basic building block with (1x1, 3x3, 1x1) convolutions. We slowly evolve the architecture choices to better understand its impact on the model performance. Key parameters include:

1) **Input Size**: The dimensions of the input images or data affect the network's computational load and memory bandwidth requirements. Larger input sizes can capture more detailed information but require more processing power, while smaller input sizes reduce computational demand at the cost of potentially losing fine-grained details. In this analysis we explore how the impact of input size ranging from 32x32 to 1024x1024 affects the performance on device due to its compute and memory bandwidth implications.

2) **In/Out Feature Size**: The input and output feature sizes determine the breadth of the feature maps at each layer. These sizes are crucial for balancing the network's capacity to learn complex features against the computational and memory resources required. The input/output feature sizes are intentionally multiples of 16 or 32 for better efficiency on the hardware, across wide range of values ranging from 4 to 256. The features map sizes are multiples for 2x, 4x, 8x, the basic width within the different modules.

3) **Grouped Convolution**: Grouped convolutions divide the input channels into smaller groups for separate processing. This reduces the number of parameters and computational complexity for efficient model training and inference. It is particularly useful in architectures like ResNeXt and MobileNet. For groups parameters in the range [4, 256], whenever the groups > input/output features, it is replaced by depth-wise convolution. We use groups = nan as a placeholder for depth-wise convolution i.e. it adopts the groups = input channels.

4) **Activations**: Activation functions introduce non-linearity into the network, enabling it to learn complex patterns.

Common activation functions include ReLU, Leaky ReLU, Softmax and Swish. The choice of activation function impacts the model's training dynamics and overall performance. The activations significantly impact the training and inference performance, with ReLU/ReLU6 activations popular due to its simplicity and hardware friendly implementation. In this analysis we will be limiting to ReLU activation, as it is popular and allows for layer fusion optimizations for model deployment.

5) **Network Depth**: The number of layers in a network defines its depth. Deeper networks can model more complex functions and hierarchical features but are prone to issues like vanishing gradients and increased computational demands. In the case of deployment on accelerators (NPU), the network depth also impacts the memory latency significantly as the activations need to be moved in/out of the local memory of the acceleration due to limited memory size.

Careful design and tuning of these parameters are essential to developing efficient and effective neural networks tailored to specific platform and its compute/memory constraints.

III. Implementation

Deploying CNN models effectively requires a robust software framework and various optimization techniques to ensure efficient performance on diverse hardware platforms. Popular frameworks such as TensorFlow, PyTorch, and ONNX provide the tools necessary for model training, evaluation, and deployment. To enhance model efficiency, optimizations like pruning and quantization are employed. Pruning reduces model size by removing redundant parameters, while quantization converts model weights and activations from floating-point to lower precision, thereby decreasing memory usage and computational requirements. Hardware accelerators such as GPUs, TPUs, and NPUs (specialized AI chips) are integral to speeding up CNN inference. These accelerators are designed to handle the parallel nature of CNN operations, delivering significant performance boosts and enabling real-time processing capabilities in applications such as image recognition and object detection.

Most of the current NAS algorithms focus on model accuracy and compute, without understanding the implications of these choices on inference or hardware accelerator efficiency. This leads to replacing compute intense operations (i.e. Convolutions) with relatively more memory intense operations (i.e. Depth-wise Convolutions), which fail to map efficiently to the hardware accelerators that are designed for CNN. This leads to degradation in performance or runtime inference time as shown in Fig 1. As the choices for different hardware accelerators varies, it is not possible to have a one size fits all solution. The model needs to understand the on-device capabilities and make decisions accordingly for efficiency. This work explores the CNN design space to help model design choices and guide the design choices. These methodologies can be applied to different platforms.

A. Edge AI Inference

The trained model is deployed on the edge device using offline tools. These deployment tools need to support different frameworks like PyTorch/Tensorflow or use a model exchange format like ONNX. For our analysis we will be using Ryzen AI software to deploy ONNX models on to the NPU (Phoenix).

The Ryzen AI software framework does the following:

i. Convert the model to format compatible with the inference platform i.e. converting from TFLite/PyTorch model to ONNX model.

ii. Optimize the model graph through layer fusion and quantization.

iii. Split the graph into sub-graphs for model execution on different hardware CPU/GPU/NPU

iv. Precompute buffer and allocation for model execution on the device

v. Mapping models Ops to hardware low-level kernels and code generation.

Layer fusion is an important graph optimization that combines adjacent operations like Conv + ReLU or Conv + BN + ReLU, to avoid additional memory latency involved to move the intermediate results. Quantization [18] is another popular technique that reduces the memory footprint and compute efficiency of CNN model by reducing the precision of weights and compute from float (FP32) to integer (INT8) operations. While it does have its own challenges in term of maintaining model accuracy, it is a popular technique due to inherent support for INT8 compute on many edge inference platforms.

B. Measurement Tool

The measurement tools measure the throughput and latency of the model deployment on hardware. The graph is converted to ONNX model, which is future optimized through quantization (i.e. INT8) for the target platform by using the ONNX quantization tool. The quantized graph is deployed on the device for latency and throughput measurement. For better analysis of hardware efficiency, we compute the M MACs/Sec, which is a normalized measure of hardware throughput. This provides more balanced view of hardware efficiency across different network models.

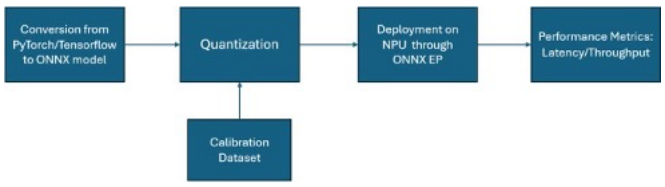

Fig. 4 Measurement tools for the inference latency/throughput

IV. Results and Analysis

In this section, we summarize the results for different network parameters and characterize the performance of the

inference framework and inference platform. These results can be guidelines for CNN design for runtime deployment. It also enables efficient model design of inference deployment on hardware. It gives useful insights into the effect of different building blocks, input size, grouped convolution operations, network width and network depth. While these results and analysis are focused on CNN due to its better understanding of design space, they can be applied to Transformers and similar architectures. These results characterize the performance of NPU for different network design choices, giving valuable insights to the AI/ML engineers about the inference framework and platform.

A. NPU Performance Characteristics

To better understand the performance characteristics of the CNN model deployment at the Edge, we try to highlight one dimension by sweeping it across the search space while keeping all the other dimension constant. This allows us to analyze the impact of the hyperparameter on the model deployment. During this analysis we try to answer the following: i) what is the impact on model performance, ii) what is the reason for this characteristic and iii) what are guidelines for implementation of NN design.

B. Input Size

Input image size has significant effect on the network accuracy and performance. As it decides the amount of computation needed and determines the intermediate activation size. Hence, it has a twofold impact on both compute and memory bandwidth. In Fig. 5, where we compare the different input size profiles for combinations of CNN models on CPU, we see a roof line curve maximizing at 224x224 resolution. There might be different factors contributing to this, such as the efficient or customized kernels might be designed for the more popular input resolutions, or the inputs hardware tradeoffs yield best results for that specific resolution. This gives AI/ML engineers the necessary information to make design choices for input resolution for optimal performance.

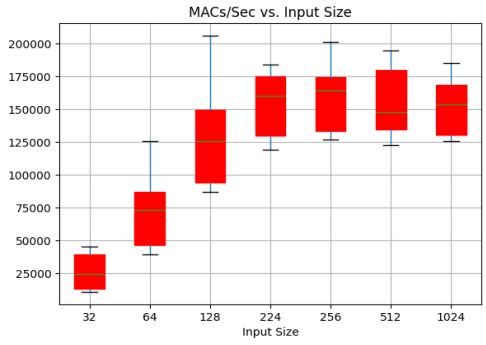

Fig. 5 Hardware throughput efficiency (M MACs/sec) vs Input Size on CPU

C. Feature Size/Width

These feature sizes determine the number of input/output feature maps at each layer a.k.a network width. The hardware accelerators are designed as SIMD machines to exploit the parallelism within the convolutions that form the bulk of network compute. These accelerators have SIMD width that are multiples of 32/64. To have the efficient implementation we see in Fig. 6 when the width is increased in multiples of 32, the overall hardware efficient is improved.

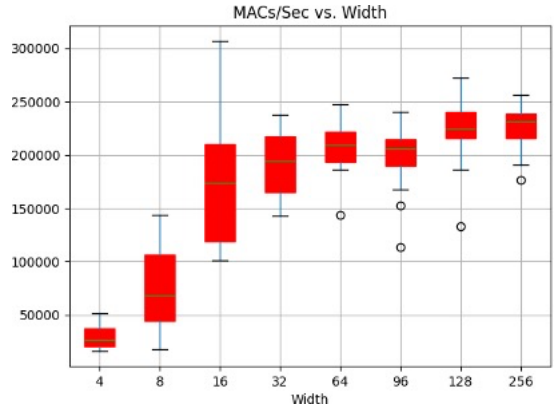

Fig. 6 Hardware throughput efficiency (M MACs/sec) vs width on CPU

Fig. 7 shows the impact of width = 32/64 on a range of CNN models. We see a significant improvement in hardware efficiency, which is even more pronounced for regular convolution compared to depth-wise convolution. In summary, instead of slowly growing the feature map size, if the network is able increase feature maps size to multiples of 32 at the earliest i.e. in the first few layers, we see a boost in the overall performance on the device.

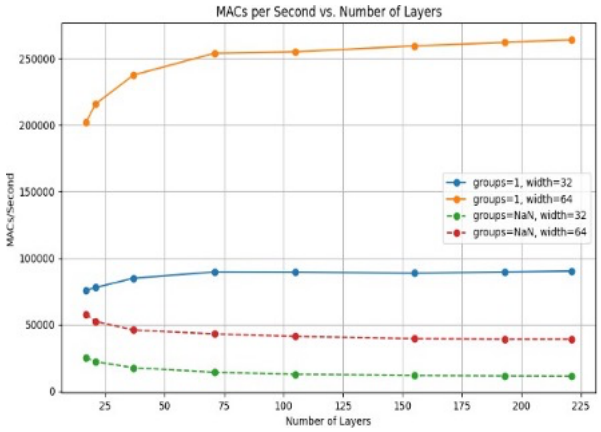

Fig. 7 Hardware throughput efficiency (M MACs/sec) vs number of layers on CPU

D. Network Depth

With the advent of efficient architectures, we see more and more depth-wise convolutions. These have only a fractional computational cost compared to regular convolutions, allowing these networks to grow much deeper for the same compute budget. However, in case of deployment on accelerators, the network depth also impacts the memory latency significantly as the activations need to be moved in/out of the local memory of the acceleration due to limited memory size. Fig. 8 shows the latency profiles of these networks across different widths. We see that convolutions (groups=1) show higher latency due to significantly larger MACs compensating for the hardware inefficiencies in the depth-wise convolutions (groups=nan). An interesting observation is how CPU is better able to handle the depth-wise convolutions across different depths compared to which shows a linear increase in latency, due to memory latency overheads for the data movement.

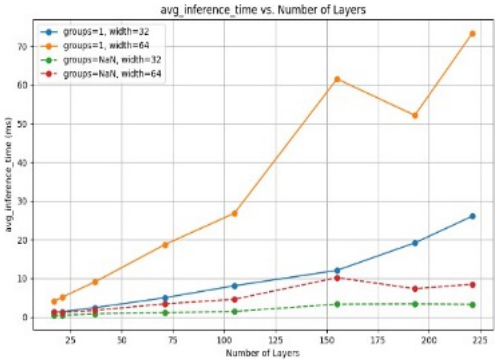

Fig. 8 Latency (ms) vs network depth on CPU

E. Grouped Convolution

Grouped convolutions is a compute efficient convolution operation used to reduce the compute intensity of the model. In Fig. 9, we can clearly see distinct characteristic profiles for grouped convolutions on CPU. In the case of CPU, we see the groups = 1,2 seems to have high hardware efficiency compared to larger groups like 16/32. As shown in the figure, as it draws closer to depth-wise convolution we again see a spike in the performance. This can be explained by the generic processor (CPU) and cached based memory system is able to efficiently handle depth-wise convolution. On the contrary we see a steady fall in efficiency, which are designed to leverage the parallelism and data reuse, which suffers from the excessive memory latency due to limited reusability of weight/inputs for depth-wise convolutions. Moreover, these bulky regular convolution with potential redundancy make them robust to quantization noise.

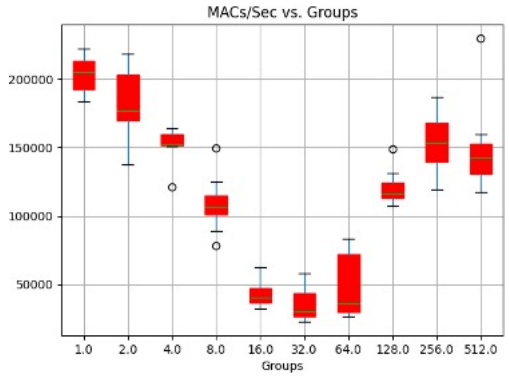

Fig. 9 Hardware throughput efficiency (M MACs/sec) vs groups on CPU

In summary, we present a comprehensive performance results in the CNN design space to learn the characteristics of inference platform. The findings provide an explanation for implication of design choices in the design space.

V. Conclusion

In conclusion, we have characterized a collection of CNN model architectures to enable efficient design choices. Our analysis provides guidelines across various parameters to enhance the efficiency of CNN models on hardware. Our findings demonstrate that hardware-aware design choices for CNNs can significantly improve overall efficiency, maximizing the capabilities of both software frameworks and hardware accelerators. This research underscores the importance of considering hardware constraints and opportunities in the design of CNN models to achieve optimal performance.

VI. Future Work

In this paper, we presented a systematic study on the impact of various macro network architecture choices in a controlled setting. While this paper primarily discusses CNN models, similar analysis can be applied to transformers or similar models for better model design. Moreover, expanding this study on larger datasets of NAS models with various network configurations will give a more comprehensive understanding of performance characteristics on a wide range of model architectures.

References

[1] Facebook, "Accelerating Facebook's Infrastructure with Application specific Hardware," https://engineering.fb.com/2019/03/14/data-center-engineering/accelerating-infrastructure/, 2021.
[2] "Edge TPU," https://cloud.google.com/edge-tpu, accessed: 2021-01-09.
[3] EETimes, "AWS Rolls Out AI Inference Chip," https://www.eetimes.com/aws-rolls-out-ai- inference-chip/, 2021.
[4] Ren, P., Xiao, Y., Chang, X., Huang, P.-y., Li, Z., Chen, X., and Wang, X. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Comput. Surv.*, 54(4), 2021.
[5] 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.

[6] Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pp. 6105-6114, 2019.

[7] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuarells and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510-4520, 2018

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2016.

[9] "RyzenAI Software": https://www.amd.com/en/developer/resources/ryzen-ai-software.html

[10] Hadjer Benmeziane et al. 2021. A comprehensive survey on hardware-aware neural architecture search. *arXiv preprint arXiv:2101.09336*, 2021.

[11] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, Cong Hao, and Yingyan Lin. 2021. HW-NAS-Bench: Hardware-Aware Neural Architecture Search Benchmark. In *Proc. Int. Conf. Learn. Represent.* https://arxiv.org/abs/2103.10584

[12] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016. [Online]. Available: http://arxiv.org/abs/1611.01578

[13] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "Nsga-net: Neural architecture search using multiobjective genetic algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019. [Online]. Available: https://doi.org/10.1145/3321707.3321729

[14] X. Zhang, Z. Huang, and N. Wang, "You only search once: Single shot neural architecture search via direct sparse optimization," CoRR, vol. abs/1811.01567, 2018. [Online]. Available: http://arxiv.org/abs/1811.01567

[15] L. Zhang, Y. Yang, Y. Jiang, W. Zhu, and Y. Liu, "Fast hardware-aware neural architecture search," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 2959-2967.

[16] Hitawala, S. Evaluating ResNeXt Model Architecture for Image Classification. *arXiv 2018, arXiv:1805.08700*

[17] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-theart," *Knowl.-Based Syst.*, vol. 212, Jan. 2021, Art. no. 106622.

[18] Dwith Chenna, "Quantization of Convolutional Neural Networks: A Practical Approach", *International Journal of Science & Engineering Development Research*, Vol.8, Issue 12, page no.181 - 192, December-2023, Available: http://www.ijrti.org/papers/IJRTI2312025.pdf