A Hardware-Centric Approach to Increase and Prune Regular Activation Sparsity in CNNs

Tim Hotfilter, Julian Hoefer, Fabian Kreß, Fabian Kempf, Leonhard Kraft, Tanja Harbaum, Jürgen Becker
Karlsruhe Institute of Technology (KIT)
{hotfilter, julian.hoefer, fabian.kress, fabian.kempf, tanja.harbaum, becker}@kit.edu

***Abstract*—A key challenge in computing convolutional neural networks (CNNs) besides the vast number of computations are the associated numerous energy-intensive transactions from main to local memory. In this paper, we present our methodical approach to maximize and prune coarse-grained regular blockwise sparsity in activation feature maps during CNN inference on dedicated dataflow architectures. Regular sparsity that fits the target accelerator, e.g., a systolic array or vector processor, allows simplified and resource inexpensive pruning compared to irregular sparsity, saving memory transactions and computations. Our threshold-based technique allows maximizing the number of regular sparse blocks in each layer. The wide range of threshold combinations that result from the close correlation between the number of sparse blocks and network accuracy can be explored automatically by our exploration tool Spex. To harness found sparse blocks for memory transaction and MAC operation reduction, we also propose Sparse-Blox, a low-overhead hardware extension for common neural network hardware accelerators. Sparse-Blox adds up to 5× less area than state-of-the-art accelerator extensions that operate on irregular sparsity. Evaluation of our blockwise pruning method with Spex on ResNet-50 and Yolo-v5s shows a reduction of up to 18.9% and 12.6% memory transfers, and 802 M (19.0%) and 1.5 G (24.3%) MAC operations with a 1% or 1 mAP accuracy drop, respectively.**

***Index Terms*—structured activation pruning, sparse DNNs**

I. Introduction

Deep Neural Networks (DNNs) have been successfully deployed in numerous areas over the last decade, from image classification, where they outperform human doctors [1], to assistive robotics [2]. However, the growing computational complexity and memory requirements of DNNs pose a serious challenge to computing devices, especially to embedded systems. To keep pace with the progress, non-Von Neumann architectures like systolic arrays or vector processors, which use dataflow computing paradigms, have established themselves. Energy savings are achieved by incorporating multiple optimization techniques like quantization and pruning [3]–[5]. However, combining high throughput and low energy consumption is still a major challenge yet to be solved. Since DNNs, especially Convolutional Neural Networks (CNNs), have large memory requirements, intermediate results are offloaded from local into off-chip memories. The largest share of energy consumption originates from this offloading [6].

Nevertheless, many CNN feature maps show large sparsity in activations [7] due to the rectifier activation function. Activation pruning exploits this sparsity, by skipping operations that do not contribute to the result. However, sparse values in activations of CNNs have a high irregularity [8]. To prune irregular sparsity, CNN accelerators have to implement complex and area expensive element indexing or compression mechanisms. In contrast, regular pruning mechanisms are cheaper and easier to implement in hardware [9].

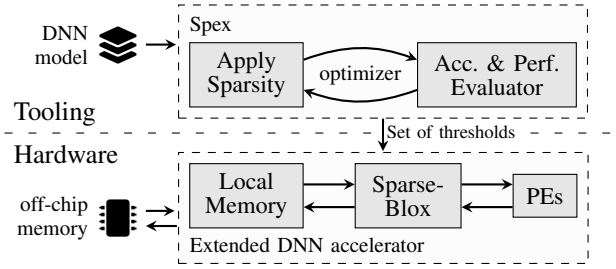


Fig. 1: Spex and Sparse-Blox: A tool and hardware extension to increase and exploit regular blockwise sparsity in CNNs.

To increase and prune regular sparsity, we propose the DNN accelerator extension Sparse-Blox and the corresponding exploration tool Spex. Our approach divides each activation feature map into blocks of a given dimension, compares each block's sum to a threshold, and then decides whether a block can be pruned. As shown in Figure 1, our hardware-aware optimization and exploration tool Spex aims to find beneficial thresholds for each layer before deployment to the hardware. Therefore, it automatically explores the search space of the correlation between pruned sparse blocks and the network accuracy. After threshold definition, the low-overhead hardware extension Sparse-Blox applies our approach to common DNN hardware accelerators. It enables pruning of entire blocks in activations. This makes compression of sparse values and skipping of MAC operations inexpensive in hardware and saves off-chip memory transfers. Our code and examples are provided to the public[1].

In summary, our contributions are threefold:

- We present the tool Spex to find beneficial thresholds that increase blockwise sparsity in activation feature maps.
- We propose the low-overhead pruning hardware extension Sparse-Blox, adding 2.5–5.0× less area compared to state-of-the-art activation pruning extensions.
- We evaluate our approach with different CNNs and exploration configurations. Exemplary, in ResNet-50 we can reduce the operations by 19% and save 8.4 M memory transfers (18.9%) with 1% accuracy drop.

[1]https://github.com/itiv-kit/dnn-model-exploration

II. Related work

Pruning of weights (W) and input activations (IA) that occurs in a structured or unstructured form has been explored in multiple studies. Zhou et al. learned regular sparse pattern to maximize the compression ratio and the performance on GPUs [10]. They divided the number of operations and parameters in a ResNet-50 by four, with only 2% accuracy drop. Zhu et al. exploit structured sparsity with their FPGA-based accelerator, showing that structured sparsity is much more hardware friendly than irregular sparsity [9], especially when weights are trained to show structured sparsity.

To efficiently utilize activation sparsity, hardware architectures have to support activation pruning. Kim et al. demonstrate a 6× speed up on VGG-16 [11] by skipping computations of sparse elements in input feature maps in hardware. However, to save memory bandwidth, compression techniques that pool zero activations have to be applied. E.g., Cnvlutin by Albericio et al. [3] compresses sparse rows of feature maps in hardware, demonstrating a 1.37× acceleration without accuracy loss. SCNN [4] uses index-based compression for significant energy and computation reduction, yielding a 2.7× performance increase, however, they have to add 33% extra area for encoding. STICKER [12] exploits W and IA sparsity through different sparse operation modes, showing a small area overhead and better performance. They report a total 5% area share for their approach and are able to reduce the size of buffers compared to reference works. SNAP [5] uses a novel associative search and compression algorithm for unstructured sparsity, demonstrating up to 3× less write back traffic and a power efficiency of 3.61 TOPS/W. Although they can benefit from many sparse computations, compression techniques usually come with hardware and scheduling overheads.

III. Overview of Sparse-Blox and Spex

Due to the fact that negative results of filters become zero after applying the commonly used Rectified Linear Unit (ReLU) activation function, up to 80% sparse activations can be found in recent CNNs [5]. Activation pruning leverages these zero values by skipping corresponding MAC operations in the next layer and compressing them to reduce memory offloading. Especially, regular sparse pattern support compression and computation reduction, e.g., using pooled sparse values the computation of an entire 8×8 block of sparse inputs on an 8×8 weight stationary systolic array can be skipped. Our hardware-centric approach enables inexpensive exploitation of blockwise structured sparsity in CNNs. The general principle of our blockwise pruning method is given in Figure 2. First, an activation feature map is divided into smaller blocks that match the target hardware accelerator. Since entire sparse blocks are uncommon in out-of-the-box CNNs, we introduce a threshold that is compared to the sum of each block to determine whether a block can be pruned. The set of thresholds achieving the best tradeoff between activation compression, computation reduction and accuracy impact is determined in advance by our exploration tool Spex. This beneficial set of thresholds is loaded into our hardware extension Sparse-Blox

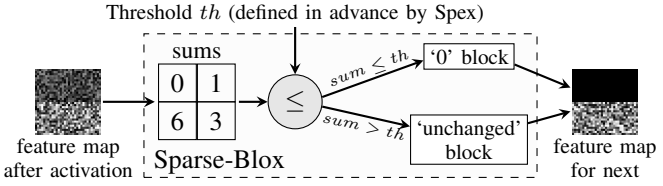


Fig. 2: Exemplary scheme of operation, how our pruning flow leverages blockwise sparsity in CNNs

that accumulates blocks and compares them with the threshold. Both operations are inexpensive to implement in hardware.

A. Sparse-Blox Exploration Framework: Spex

Spex explores the design space of threshold combinations for the best trade-off between created blocks and algorithm accuracy. Figure 3 gives an overview of Spex. It takes a workload description which contains a pretrained neural network reference model, the validation dataset required for accuracy computation, and a description of the target hardware, i.e., the block size. The centerpiece of Spex is a Genetic Algorithm (GA) that systematically generates sets of thresholds. These sets are then evaluated in a modified model implementing Sparse-Blox's blockwise pruning flow. For iterative optimization, the resulting accuracy is fed back.

Spex is designed extendable and can work with any PyTorch model or dataset. To evaluate the trade-off between increased blockwise sparsity and model accuracy, Spex alters the PyTorch model by replacing layers with counterparts that apply our blockwise pruning flow. Each layer is annotated with a threshold value taken from the genome. To reflect the way the inference is computed in hardware, we apply the im2col transformation to convolutional layers to get a 2D matrix. Fully connected layers do not require any further modification. Other layer types might be added as well. During inference, the modified forward function divides the activation matrix into blocks, computes the sum of each, compares it with the individual threshold, and sets them either to zero or leaves them untouched. Each position of a changed block as well as an average of the sum is stored for later evaluation. To speed up the inference, Spex supports parallel execution on multiple threads and HPC computing nodes.

As exploration algorithm, we choose the well-established NSGA-II [13] from the Pymoo library [14] to unify adjustability and performance on multi-objective problems. The GA's evaluation function tries to maximize both the created sparse blocks and the validation accuracy. To determine the accuracy, we can work with different evaluation metrics, e.g., Top-1 classification accuracy, intersection over union for segmentation tasks or box average precision for object detection. A lower accuracy constraint may be added to drop solutions that do not meet the design requirements. Spex stops the GA, when a feasible number of solutions is found. With the obtained solutions, our tool generates various metrics to reveal insights of the investigated CNN on the given hardware configuration, like the Pareto optimal solutions or the threshold distribution.

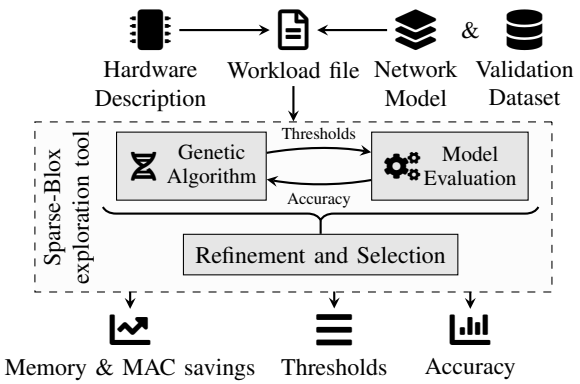


Fig. 3: Components and overview of Spex

B. Hardware Accelerator Extension: Sparse-Blox

The advantage of our sparsity approach becomes apparent when it is integrated in a low-power CNN hardware accelerator for embedded vision applications. Common accelerators like systolic arrays leverage spatially independent computations in CNNs by connecting hundreds of Processing Elements (PEs) in a 2D mesh to reuse weights and input with neighboring PEs. To utilize systolic arrays for inference, large matrices of state-of-the-art CNNs have to be broken down into smaller blocks. While processing these smaller blocks, our proposed blockwise pruning scheme comes in.

Our hardware accelerator extension Sparse-Blox monitors the magnitude of incoming activations and prunes an entire block based on the previously determined threshold. Sparse-Blox is inserted in between the data path of the PEs array to the local memory, as shown in Figure 4. We only have to add three components, while the other common parts of a CNN accelerator are left untouched. The results of the PEs are accumulated by an adder tree and then compared to the configured threshold. If the sum is less than the threshold, pruning happens. Now the result can be discarded and a *'0'-entry* is stored in a dedicated cache and not in the local memory. The cache entry represents the address of a sparse block and requires only a fraction of the memory compared to a whole block stored in local memory. Resulting free memory can be used to reduce offloading of results or to prefetch the next operands. If the result is above the threshold, matrix results are stored to the local memory in the usual way. Skipping the computation of a whole block happens as soon as the PEs request data that is present in the dedicated cache. On a cache hit, an entire block result can be defaulted to zero. On a cache miss, the computation runs without modification.

IV. Evaluation and Discussion

We perform extensive experiments with Spex to understand the impact of various parameters on different workloads. This understanding is crucial when activation pruning is ultimately exploited later in the hardware accelerator. To cover various hardware architectures, we look at block sizes of 8×8, 16×16 and 1×16 to represent embedded systolic arrays as well as

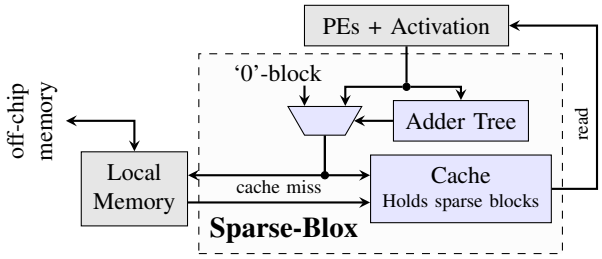


Fig. 4: Envisaged Sparse-Blox hardware extension

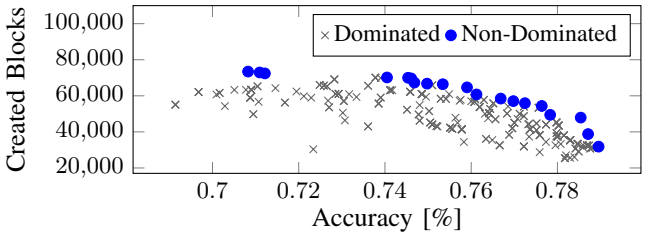
vector processors. As workloads we choose ResNet-50 [15] trained on ImageNet-1K [16] achieving a Top-1 classification accuracy of 80.85% and Yolo-v5s [17] trained on COCO [18] achieving a detection precision of 37.4 $mAP_{val}(50:95)$. They represent demanding tasks that occur in embedded vision, e.g., autonomous driving, and are ideal candidates for optimization through Spex and Sparse-Blox to enable application in energy and resource constrained embedded environments. Before deploying on the target accelerator, we analyzed over 100 individual configurations to determine the best set of thresholds for each accelerator and workload.

A. Algorithm Constraints & Evaluation Setup

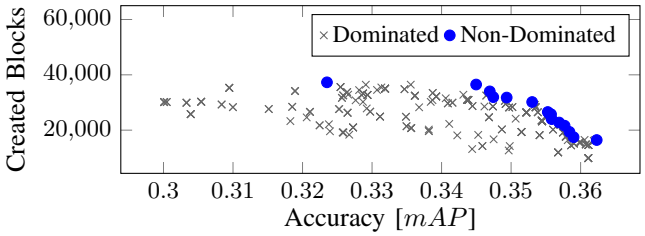
Since CNN accuracy decreases rapidly with too high threshold values that prune too many features, we constrain the exploration with an upper threshold limit that we determine based on the distribution of activations. In all experiments, the GA evaluates a population of 15 individuals over 25 generations, to equalize runtime and number of distinct solutions. To allow the GA to break out of the initial population fast and converge, we test different GA parameters and choose a mutation factor of 30 and a crossover factor of 20 with a probability of 0.9 for both.

B. Amount of Sparse Blocks and Savings during Inference

The most relevant finding of Spex's exploration is the amount of sparse blocks created with a given set of thresholds and how this impacts the accuracy. More created sparse blocks allow skipping more computations and are saving more memory transfers during inference. For ResNet-50 and Yolo-v5s, all feasible solutions and the Pareto front are shown in Figure 5a and Figure 5b, respectively. Depending on the application's requirements, these solutions have to be evaluated regarding acceptable accuracy degradation and leveraged sparsity. The found correlation between created sparse blocks and the drop in accuracy is not linear, but there are sweet spots. For example, in the commonly used image classification CNN ResNet-50, which has in total 345,400 8×8 block per inference, Spex is able to create 51,642 (14.9%) new 8×8 sparse blocks with a usually acceptable accuracy drop of 1%. Together with the already present sparse blocks, we can save 8.42 M or 18.9% memory transfers and 802 M or 19.04% MAC operations in ResNet-50. For Yolo-v5s, with a total of 156,880 8×8 blocks, 30,173 or 19.23% sparse blocks are created with an accuracy drop of 1 mAP. In combination



(a) Solutions for ResNet-50 with block size 8x8.



(b) Solutions for Yolo-v5s with block size 8x8.

Fig. 5: Pareto optimal solutions, showing accuracy over the amount of sparse blocks created.

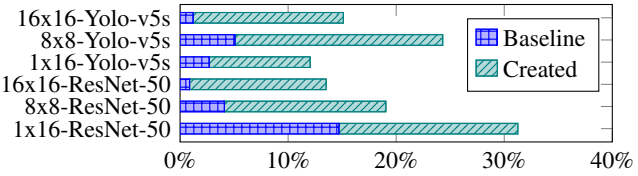


Fig. 6: Percentage of sparse blocks before and after sparsity maximization through Spex with 1% accuracy drop.

with the already present sparsity, this saves in total 4.88 M or 12.6% memory transfers and 1.5 G or 24.31% MACs.

Besides 8×8 blocks, we investigate other block sizes, shown in Figure 6. In general, smaller block sizes, e.g., 1×16 for vector processors, result in more sparse blocks since feature maps divided into smaller blocks store less information in a block and hence are more robust to pruning. For example, with 1×16 blocks in ResNet-50, we can prune over 30% of all operations. Also, the number of created sparse blocks does not scale linearly. For example, in ResNet-50 with 1% accuracy drop, 11,133 or 12.64% sparse 16×16 blocks are created, but only 231,688 or 16.56% 1×16 blocks.

C. Comparison to state-of-the-art Sparse DNN Accelerators

The claim of our approach is a straightforward integration into common hardware accelerators while adding only little area. To prove this claim, we show an area overhead comparison of Sparse-Blox to other CNN accelerators working on activation sparsity in Table I. The reference overhead is taken directly from the respective paper. To allow for a good comparison, we added the components of Sparse-Blox, a cache, an adder tree and the corresponding decision logic to the reference accelerators. Since none of them are open-sourced, we model each accelerator in Accelergy [19]. We match the size of sparse blocks to the organization of PEs in the reference accelerators.

TABLE I: Area overhead comparison of Sparse-Blox against other state-of-the-art sparse DNN accelerators

	SCNN [4]	STICKER [12]	SNAP [5]
Sparsity module			
Reference	24.8 %	4.76%	12.5 %
Sparse-Blox	4.95%	1.73%	4.81%
Overhead reduction	5.01×	2.75×	2.59×

For the most recent works STICKER [12] and SNAP [5], we can show an area overhead reduction of about 2.5×, and for SCNN [4] even up to 5×. Since all three accelerators work on different PE array dimensions and feature different sizes of on-chip memories, which largely contribute to the area, we give the relative numbers in the table for better comparison. With SCNN's 4×4 PE array, for example, we are able to find a threshold combination that leverages 348,764 sparse blocks in ResNet-50, resulting in 26.6% pruned sparsity with only 1% accuracy degradation. Considering the 3×7 PE arrays of SNAP, about 312,000 or 24.1% of all blocks in ResNet-50 can be pruned. Looking at the area breakdown, the largest portion of Sparse-Blox accounts for the cache, since it has to hold all sparse block positions from one to another layer. To have sufficient space the cache of our hardware extensions for SNAP, STICKER and SCNN, can store up to 6,144, 512 and 8,196 16-bit block positions, respectively. Even if regular activation sparsity may exploit slightly less sparsity compared to irregular, comparing our approach to state-of-the-art DNN accelerators, we can demonstrate a reduction in the number of computations and memory transfers by on average 25%, but with 5× less area overhead.

V. Conclusion

In this paper, we presented our hardware-centric approach to increase regular blockwise activation sparsity in CNNs. Therefore, we first introduced Spex an exploration tool to analyze the CNN workload and to maximize regular sparsity, and secondly Sparse-Blox, a low-overhead accelerator extension to enable pruning of sparse activation feature maps. Since pruning has an impact on the accuracy, Spex evaluates the large number of suitable threshold combinations in state-of-the-art CNNs. Extensive experiments on CNN benchmarks, show that we can reduce memory transfers and MAC operations by around 30%. We compared Sparse-Blox with state-of-the-art CNN accelerators, showing a 2.5–5× smaller area overhead. With our demonstrated optimizations, we support vision tasks in resource constrained embedded environments. Finally, we made Sparse-Blox open-source for further research.

Acknowledgment

This project has received funding from the European High Performance Computing Joint Undertaking (JU) under Framework Partnership Agreement No 800928 and Specific Grant Agreement No 101036168 (EPI SGA2).

REFERENCES

[1] Yunzhu Li, Andre Esteva, Brett Kuprel, Rob Novoa, Justin Ko, and Sebastian Thrun. Skin cancer detection and tracking using data synthesis and deep learning. Dec 2016. arXiv: 1612.01074.

[2] Iris Walter, Jonas Ney, Tim Hotfilter, Vladimir Rybalkin, Julian Hoefer, Norbert Wehn, and Jürgen Becker. Embedded face recognition for personalized services in the assistive robotics. In *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, Communications in Computer and Information Science, page 339–350, Cham, 2021. Springer International Publishing.

[3] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, page 1–13, Jun 2016.

[4] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 45(2):27–40, Jun 2017.

[5] Jie-Fang Zhang, Ching-En Lee, Chester Liu, Yakun Sophia Shao, Stephen W. Keckler, and Zhengya Zhang. Snap: An efficient sparse neural acceleration processor for unstructured sparse deep neural network inference. *IEEE Journal of Solid-State Circuits*, 56(2):636–647, Feb 2021.

[6] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, page 10–14, Feb 2014.

[7] Nanfei Jiang, Xu Zhao, Chaoyang Zhao, Yongqi An, Ming Tang, and Jinqiao Wang. *Pruning-aware Sparse Regularization for Network Pruning*. Number arXiv:2201.06776. Jan 2022.

[8] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. (arXiv:1902.09574), Feb 2019.

[9] Chaoyang Zhu, Kejie Huang, Shuyuan Yang, Ziqi Zhu, Hejia Zhang, and Haibin Shen. An efficient hardware accelerator for structured sparse convolutional neural networks on fpgas. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(9):1953–1965, Sep 2020.

[10] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. *Learning N:M Fine-grained Structured Sparse Neural Networks From Scratch*. Number arXiv:2102.04010. Apr 2021.

[11] Dongyoung Kim, Junwhan Ahn, and Sungjoo Yoo. Zena: Zero-aware neural network accelerator. *IEEE Design Test*, 35(1):39–46, Feb 2018.

[12] Zhe Yuan, Yongpan Liu, Jinshan Yue, Yixiong Yang, Jingyu Wang, Xiaoyu Feng, Jian Zhao, Xueqing Li, and Huazhong Yang. Sticker: An energy-efficient multi-sparsity compatible accelerator for convolutional neural networks in 65-nm cmos. *IEEE Journal of Solid-State Circuits*, 55(2):465–477, Feb 2020.

[13] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[14] J. Blank and K. Deb. pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509, 2020.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 770–778, Las Vegas, NV, USA, Jun 2016. IEEE.

[16] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. (arXiv:1409.0575), Jan 2015.

[17] Ultralytics. Github - ultralytics/yolov5, 2022.

[18] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. *Microsoft COCO: Common Objects in Context*. Number arXiv:1405.0312. Feb 2015.

[19] Y. N. Wu, J. S. Emer, and V. Sze. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.