NNBench-X: A Benchmarking Methodology for Neural Network Accelerator Designs*

Xinfeng Xie, Xing Hu, Peng Gu, Shuangchen Li, Yu Ji, and Yuan Xie
University of California, Santa Barbara
Santa Barbara, California, U.S.A

Abstract

The tremendous impact of deep learning algorithms over a wide range of application domains has encouraged a surge of neural network (NN) accelerator research. Prior NN accelerator research reveals that the software-hardware co-design method is promising to achieve significant performance improvement and energy savings. To guide future co-designs, an evolving benchmark suite and its associated benchmark method are needed to characterize NN workloads and incorporate emerging NN compression techniques. However, the co-design method has not been well covered by existing NN benchmark work. In this paper, we propose a novel benchmarking methodology, which decouples the benchmarking process into three stages. First, we select the most representative applications from a user-customized candidate pool by analyzing their performance features. Second, we compress selected applications according to customized model compression techniques to generate a benchmark suite. Finally, we evaluate a variety of accelerator designs on the generated benchmark suite. To demonstrate the effectiveness of our benchmarking methodology, we conduct a case study of designing a general NN benchmark from TensorFlow Model Zoo and with various NN compression methods. We finally evaluate the benchmark on various representative architectures.

1 Introduction

Neural Network (NN) algorithms have demonstrated better accuracy than other traditional machine learning algorithms in a wide range of application domains, such as the computer vision (CV) [3, 5, 6, 11, 14] and natural language processing (NLP) [9, 12]. These breakthroughs indicate a promising future for their real-world deployment. Deploying these applications, especially for the inference stage, requires high performance under stringent power budgets, which boosts the emergence of accelerator designs for these applications. However, designing such an NN accelerator using application specific integrated circuits (ASICs) is challenging because NN applications are changing rapidly to support new functionalities and improve accuracies, while ASIC design requires a long design and manufacturing period. The accelerator design could be prone to becoming obsolete if the design fails to capture key characteristics of emerging models. Therefore, a benchmark to capture these workload characteristics is crucial to guiding NN accelerator design.

There exists plenty of NN benchmark studies. Among them, Fathom [1] and BenchIP [15] serve a similar purpose as the benchmarks we target to generate, i.e., using a benchmark suite to guide the hardware design. However, existing work will suffer from a few problems. First, the benchmark suite is selected not by any quantitative analysis. Though they justify their selection afterwards, there is a risk that their suite is not the most representative collection. Second, existing benchmarks are difficult to update or customize. It is unclear if these benchmarks are still effective for emerging algorithms. Also, they are designed for general purpose NN accelerators. NN hardware specialized for a certain application scenario (e.g., smart camera) would find it difficult to adopt these benchmarks. Third, software-hardware co-design methodology, i.e., NN model compression techniques, such as quantization and pruning, has not been adequately addressed.

In this paper we propose an end-to-end benchmarking approach for software-hardware co-designs by decoupling workload characterizations, software-level model compression strategies, and hardware-level accelerator evaluations. Our approach consists of three stages. In the first stage, *application set selection*, we characterize NN applications of interests without considering any software optimization techniques. After gathering their performance features, we select representative applications for the original application set. In the second stage, *benchmark suite generation*, users can customize the final benchmark suite according to their model compression strategies. New NN models for each application in the original benchmark suite will be generated according to software-level optimizations, such as quantizing and pruning techniques. In the last stage, *hardware evaluation*, users can provide the performance models of their accelerator designs together with the assumptions of interconnection and host. Accelerators are evaluated with the benchmark suite generated from the second stage. Power, performance, and area results are derived according to input architecture models.

To demonstrate the rationality of our benchmark, we conduct a case study on designing NN accelerators for general NN applications. For the first time, we analyze 57 models

*Our prior work [17] focuses on the characterization of NN workloads while in this paper we propose an end-to-end benchmarking method. Our prior work is the first stage of the end-to-end method presented in this paper.

EMC2'19, February 17, 2019, Washington, DC, USA
2019.

with 224,563 operators from the TensorFlow (TF) Model Zoo [4]. We generate benchmark suites by using several state-of-the-art software-level optimizations including quantizing and pruning NN models. Finally, we evaluate representative accelerators, including general purpose processors (CPU and GPU), accelerator architecture (DianNao [2]), near-data-processing architecture (Neurocube [8]), and sparse-aware architecture (Cambricon-X [18]).

Our contributions can be summarized as follows.

- We propose a novel benchmarking method, which selects the benchmark by analyzing a user-input candidate application pool, and covers software-hardware co-design configurations with high flexibility.
- We show a case study of generating a general purpose NN benchmark suite from the TF Model Zoo while applying state-of-the-art software-hardware co-designs methods, and evaluate it on representative architectures.
- We reveals insightful architecture design guidelines through our case study from the extensive profiling and analysis of a large group of NN models (57 models with 224,563 operators from the TF Model Zoo).

The rest of this paper is organized as follows. Section 2 justifies the necessity and uniqueness of our benchmarking method. Section 3 introduces our benchmark method in details. Section 4 conducts a case study on TF Model Zoo and various software-hardware co-designs. Finally, Section 5 concludes this paper.

2 Motivation: Why Another Benchmark?

Serving a similar purpose as our work, Fathom [1] is a benchmark suite composed of diverse and representative deep learning applications with detailed performance analysis. BenchIP [15] is another similar benchmark, including both a macro-benchmark suite composed of different applications and a micro-benchmark suite composed of frequently used tensor operators. *Different from them, our benchmarking methodology targets at capturing end-to-end application-to-hardware characters to guide architecture design for state-of-the-art NN workloads.* Compared with Fathom and BenchIP, our benchmark method is superior in two main aspects.

Flexible with updates and customizations: We propose a benchmarking methodology, **not** simply a benchmark suite. By doing this, we are subject to updates due to the rapid developing NN algorithms. Statics [15] have shown that within one year, the NN models proposed in top tier conference doubles. For a fixed benchmark suite, it is difficult to know whether we need to extend the suite when a new model appears,. In addition, most of the accelerators target a certain application scenario (e.g., autonomous car), instead of a general NN processor. A single one-for-all benchmark suite does not adequately address these needs. Instead, we generate different suites according to the user customized candidate application pool.

SW/HW co-design: Our benchmark method is the first for accelerators with a comprehensive awareness of software-hardware co-designs. Although BenchIP [15] includes sparse models, such as Sparse VGG, into their application set as representative workloads, these considerations are insufficient due to two reasons. First, pruned models are very similar to their original models in their work. For example, Sparse VGG performs very similar to VGG in terms of extracted performance features, making it redundant. Second, their sparsity benchmark is impossible for considering all model compression techniques. For example, structural sparsity is not covered.

3 Benchmarking Methodology

An overview of our benchmarking method is shown in Figure 1a. The rest of this section will introduce the three stages of our benchmark method in detail.

3.1 Application Set Selection

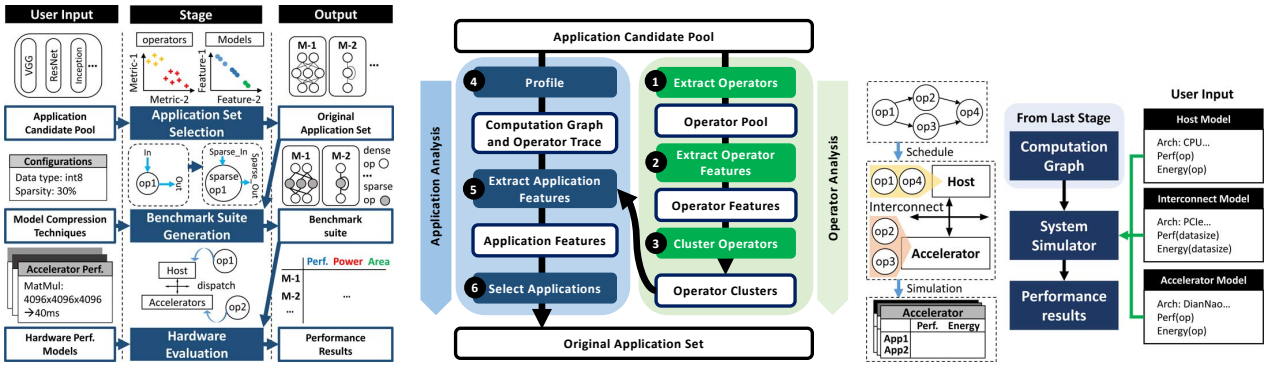
In the first stage, application set selection, we select diverse and representative NN applications out of an application candidate pool which includes the applications of user's interests. The process of this stage is shown in Figure 1b. To understand the similarity among these applications, we define the performance feature of an NN application as the time breakdown on different classes of operators. We conduct operator-level analysis and cluster operators from the application candidate pool. Then, we extract application features according to operator clusters and the profiling results of time breakdowns on different classes of operators. Finally, the original application set composed of diverse and representative workloads can be selected according to extracted application features.

Operator-level Analysis: Our operator-level analysis first extracts all operators from applications in the application candidate pool. Then, we analyze operator features key to the architecture designs. Finally, we cluster these operators. The performance feature of an operator is depicted from the following two perspectives.

Locality: This metric is defined as the amount of data needed by an operator divided by the number of scalar arithmetic computations it needs. The amount of data needed by an operator is equal to the sum of input tensor size and output tensor size. This metric reflects both the temporal locality and spatial locality in an ideal memory system where a cache hit will occur if the same location was accessed before. The lower this metric is, the better locality this operator has.

Parallelism: This metric is defined as the degree of parallelism in scalar arithmetic computations regardless of hardware resources. This metric reflects the parallelism of computations in terms of data dependency. The higher this metric is, the higher parallelism this operator has.

Application-level Analysis: We define the performance feature of an application as the time breakdown on different operator clusters obtained by the operator-level analysis. We denote the number of operator clusters as n. Specifically, the



(a) Overview. **(b)** Application set selection process. **(c)** Hardware evaluation.

Figure 1. Benchmark method.

performance feature is denoted as $\vec{f} = (R_1, R_2, ..., R_n)$ where R_i represents the percentage of the elapsed time spent in the i-th class operators. We profile each application from the application candidate pool on the baseline hardware, either CPU or GPU, to obtain its time spent in each operator class.

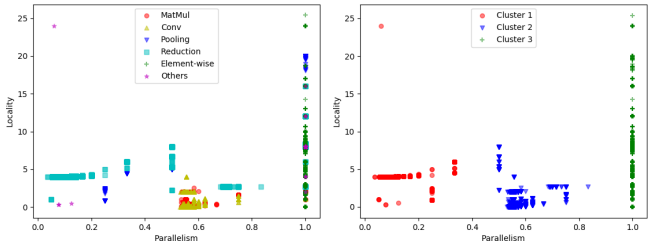
After these two level analysis, we select representative applications out of the application candidate pool to build the original application set.

3.2 Benchmark Suite Generation

In the second stage, benchmark suite generation, we provide interfaces for users to customize their NN compression techniques to generate the final benchmark suite. Each application from the original application set is a computation graph. To customize different NN model compression techniques, we provide interfaces for the users to specify the data type of tensors in this computation graph. Users can define and import model compression methods, and change the information of operators and tensors to generate the final benchmark suite according to their software-level studies in the training stage. At the end of this stage, we obtain the final test set of applications composed of quantized and pruned NN models for evaluations.

3.3 Hardware Evaluation

In the final stage, the hardware evaluation, we evaluate the generated benchmark suite on accelerator designs. Although this stage can be completed by users with detailed simulation results of accelerators, we build a system-level simulator for fast performance estimation in the initial architecture design stages to provide high-level guidelines for accelerator designs. Our system-level simulator evaluates accelerators on the generated benchmark suite by using the performance models of the accelerator, the host, and the interconnection between the accelerator and the host. The inputs and outputs of our system-level simulator are shown in Figure 1c. For each application in the generated benchmark suite, our simulator schedules operators into either the accelerator or the host by a first-come-first-serve scheduling algorithm. When an operator is not supported by the accelerator, it will be launched into the host with subsequent data transfer between the accelerator and the host.



(a) Operators from the pool. **(b)** Operator clusters.

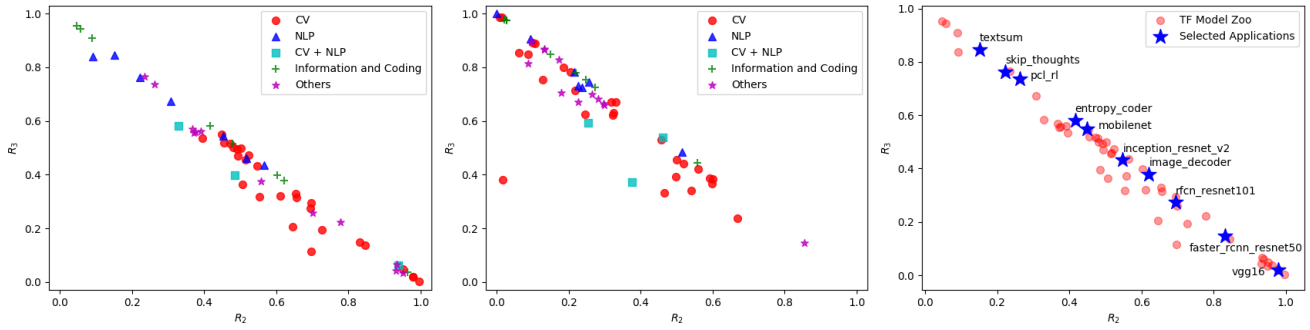
Figure 2. The distribution of operator performance features.

4 Case Study: From TensorFlow Model Zoo to a Benchmark Suite

To demonstrate the usage of our benchmark method, we introduce a case study of benchmarking general NN inference applications. To this end, we set the TensorFlow (TF) Model Zoo [4] (with 57 NN models and 224,563 operators) as the application candidate pool, and hence conduct an extensive study. This section follows the three-step process introduced in Section 3. In addition, we conclude several observations and architecture design guidelines from these case studies and show the superior of our methodology.

4.1 Application Set Selection

As the first step of our analysis flow, we apply the **operator-level analysis** to all applications from TensorFlow Model Zoo [4]. The result distribution of operator features is shown in Figure 2a. We use the k-means method in this case study resulting in three operator clusters which are shown in Figure 2b. Then, we get into the **application-level analysis** part. The application performance feature in this case study is denoted as $\vec{f} = (R_1, R_2, R_3)$, where R_1, R_2, and R_3 represent the time breakdown of an application into three operator clusters. The performance feature distributions measured on CPU and GPU are shown as Figure 3a and 3b,where x-axis stands for R_2, y-axis stands for R_3, and $R_1 = 1 - R_2 - R_3$ because $R_1 + R_2 + R_3 = 1$. Finally, we select ten applications



(a) CPU **(b)** GPU **(c)** Selected applications

Figure 3. The distribution of application features.

Table 1. Brief descriptions for ten applications selected into the original application set.

Application	Description
textsum [12]	Text summarization
skip_thoughts [9]	Sentence-to-vector encoder
pcl_rl [10]	Reinforcement learning
entropy_coder [7]	Image file compression
mobilenet [6]	Image classification
inception_resnet_v2 [5, 14]	Image classification
image_decoder [16]	Image file decompression
rfcn_resnet101 [3]	Object detection
faster_rcnn_resnet50 [11]	Object detection
vgg16 [13]	Image classification

with features evenly distributed along the line $R_2 + R_3 = 1$ to come up with the benchmark suite, NNBench-X. The distribution of these ten applications are shown in Figure 3c. Brief descriptions for these ten applications can be found in Table 1.

Insights of the analysis during benchmarking. We make several observations from the operator-level analysis (Figure 2a-2b). First, convolution and matrix multiplication operators are similar to each other, and most of them have good localities with moderate parallelism for scalar arithmetic computations. Second, all element-wise operators are the same in terms of the parallelism while the computation intensity on each tensor element can vary significantly. Third, *operators with the same or similar functionalities can have very different performance features*, such as reduction and pooling operators.

For the application-level analysis in Figure 3a-3b, we summarize the following insights. First, *Conv*, *MatMul*, and *Element-wise* operators take up a majority of the application time in most of the applications, since most of the applications distribute near the line $R_2 + R_3 = 1$. Second, comparing GPU with CPU, GPU is more likely to be bounded by R_1, due to its more powerful computing resource and higher memory bandwidth. In addition, R_3 takes a larger percentage on GPU, *indicating there are opportunities for GPU memory system optimization*. Third, we have more findings when considering the application scenarios. Most CV applications are bounded by operations from R_2 (mostly *Conv* and *MatMul*). *On the contrary, most NLP applications are bounded by operations from the R_3 (mostly element-wise operators). This indicates that memory centric computing architectures can be helpful for these NLP applications.*

4.2 Software-Hardware Co-design Evaluation

We need the benchmark generation step (Section 3.2) after application selection, in order to plug-in the NN compression setup. This step is user customized. According to our evaluation target, we generate our benchmark suite with three configurations: no compression (for GPU), quantized 16-bit fixed-point (for DianNao), and 16-bit fixed-point quantized and 90%/95% pruned (for Cambricon-X). Finally, according to performance results presented in the original papers, we derive an analytical model based on the Roofline model to estimate the performance of each supported tensor operators on accelerators. Results on the GPU are profiled and measured from the execution on a real machine. Our simulation results are shown in Figure 4.

Insights from the result. We make the following observations from Figure 4. First, GPU can benefit these applications with a higher R_2 ratio in their performance features. These applications are usually computation bound. Since applications on the x-axis are ordered by the increasing order of R_2, applications closer to the right direction along x-axis spend more time in the second cluster operators, of which most are convolution and matrix multiplication operations. As shown in Figure 4a, GPU obtains higher speedups on applications in the right side of x-axis. Second, near-data computing architectures favors applications (mostly NLP related) with a higher R_3 ratio. Figure 4b shows that Neurocube achieves higher speedups on applications on the left side of x-axis. Finally, *we found that weight pruning is less attractive for NLP applications than it is for CV applications.* Figure 4c shows comparison of DianNao and Cambricon-X in terms of performance benefits from pruning NN model weights.Comparing Cambricon-X (90%) to DianNao, Cambricon-X benefits from the reduction of computation and memory workloads due to pruned models. The results of models with different sparsities, Cambricon-X (90%) and Cambricon-X (95%), indicate that pruning more weights can have slight benefits on memory bound applications while significant benefits on computation bound applications.

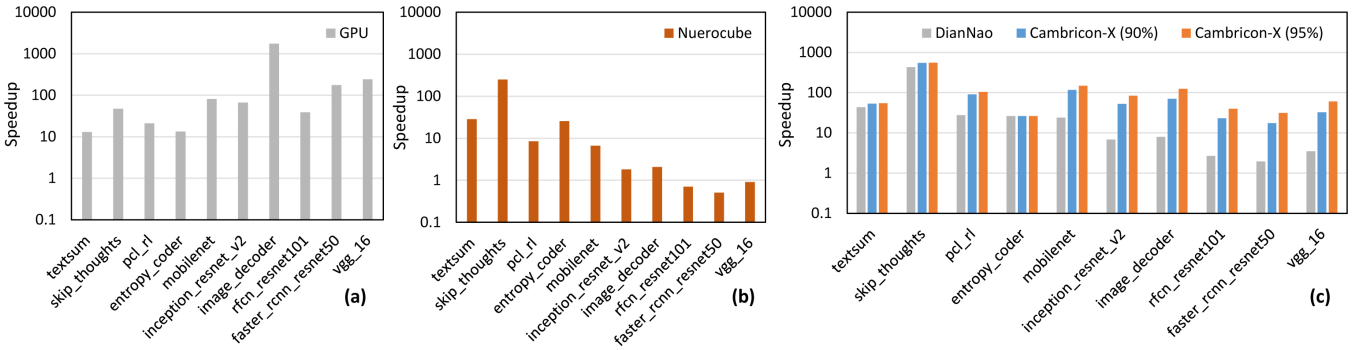


Figure 4. The speedups over CPU baseline of applications on (a) GPU without any model compression (b) Neurocube with models quantized into 16-bit fixed-point (c) DianNao with models quantized into 16-bit fixed-point, Cambricon-X (90%) with models further pruned 90% weights, and Cambricon-X (95%) with models further pruned 95% weights.

5 Conclusion

In conclusion, we propose a novel benchmarking method for NN accelerator designs. To evaluate software-hardware co-designs, our benchmark method is composed of three stages: application set selection, benchmark suite generation, and hardware evaluation. To demonstrate the usage of our benchmark method, we conduct a case study of designing NN accelerators for general NN applications. We analyze applications from the TensorFlow Model Zoo and observe that applications from the same application domains have similar bottlenecks. Moreover, we study several state-of-the-art software-hardware co-designs, including accelerator designs for quantized and pruned NN models. From our case studies, we observe that computation-centric and memory-centric architectures can have different benefits for different application domains. Also, we find that pruning NN models provides little benefit on memory-bound applications. Through our case studies and observations, we are convinced that our benchmark method is practical and feasible to provide insightful guidance to NN accelerator designs.

References

[1] Robert Adolf, Saketh Rama, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2016. Fathom: reference workloads for modern deep learning methods. In *Workload Characterization (IISWC), 2016 IEEE International Symposium on*. IEEE, 1–10.

[2] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 269–284.

[3] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 580–587.

[4] Google. 2018. TensorFlow Models. https://github.com/tensorflow/models. (2018). https://github.com/tensorflow/models

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[6] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[7] Nick Johnston, Damien Vincent, David Minnen, Michele Covell, Saurabh Singh, Troy Chinen, Sung Jin Hwang, Joel Shor, and George Toderici. 2017. Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks. *arXiv preprint arXiv:1703.10114* (2017).

[8] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. 2016. Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 380–392.

[9] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Advances in neural information processing systems*. 3294–3302.

[10] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. 2017. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems*. 2772–2782.

[11] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*. 91–99.

[12] Alexander M Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685* (2015).

[13] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[15] Jin-Hua Tao, Zi-Dong Du, Qi Guo, Hui-Ying Lan, Lei Zhang, Sheng-Yuan Zhou, Ling-Jie Xu, Cong Liu, Hai-Feng Liu, Shan Tang, Allen Rush, Willian Chen, Shao-Li Liu, Yun-Ji Chen, and Tian-Shi Chen. 2018. BenchIP: Benchmarking Intelligence Processors. *Journal of Computer Science and Technology* 33, 1 (2018), 1–23.

[16] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. 2016. Full resolution image compression with recurrent neural networks. *arXiv preprint* (2016).

[17] X. Xie, X. Hu, P. Gu, S. Li, Y. Ji, and Y. Xie. 2019. NNBench-X: Benchmarking and Understanding Neural Network Workloads for Accelerator Designs. *IEEE Computer Architecture Letters* (2019), 1–1. https://doi.org/10.1109/LCA.2019.2898196

[18] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-X: An accelerator for sparse neural networks. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–12.