2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)

Rethinking Energy-Efficiency of Heterogeneous Computing for CNN-Based Mobile Applications

Zhen Wang, Xi Li, Chao Wang, Zhinan Cheng, Jiachen Song, and Xuehai Zhou
School of Computer Science and Technology
University of Science and Technology of China, Hefei, China 230027
Email: {wzzju, chinam, jchsong}@mail.ustc.edu.cn {llxx, cswang, xhzhou}@ustc.edu.cn

***Abstract*—Convolutional Neural Networks (CNNs) have become more and more powerful in the computer vision domain, as they achieve the state-of-the-art accuracy. Despite this, it is generally difficult to apply CNNs on mobile platforms. Client-server paradigm is a straightforward way to deploy CNNs on mobile phones, but studies have shown that it suffers serious problems, such as privacy leaks. Recently, researchers focus on using heterogeneous local processors (e.g., GPUs, CPUs) to accelerate the inference of CNNs. Utilizing all local processors available can achieve the highest performance, but it might incur energy-inefficiency. Different from previous works, this paper concerns more about energy-efficiency of CNN-based mobile applications. We present an adaptive strategy, which is able to compute the energy-efficiency of all local processors, and further to obtain the energy-efficient device processor combination to perform CNN inference in parallel. The strategy is implemented on ODROID platform, where the evaluation results show that our proposed approach provides 3.67× higher energy-efficiency with only 9.7% performance degradation on average compared with the greedy strategy which tries to use all local processors available.**

1. Introduction

During the last few years, Convolutional Neural Networks (CNNs) have achieved high accuracy and strong self-adaptiveness in the field of computer vision. Many projects begin to apply CNNs on mobile platforms [1], [2], however, they often incur heavy computational overhead and power consumption. To overcome these problems, client-server computing paradigm that uses powerful clouds to perform the inference becomes the most common way. However, this approach has considerable drawbacks [3], [4] and can be summarized as follows: (1) It may leak user privacy because it needs to upload users' data to clouds. (2) Its performance is dependent on unstable and unpredictable network quality of service, such as delay, throughput, etc. (3) Some applications may upload pictures or videos to clouds frequently, and users may not accept this behavior for the mobile network has the characteristic of charge in quantity.

For all reasons above, many researchers begin to explore methods of operating inference on local device processors (e.g., GPUs, CPUs). DeepSense [4] is a mobile GPU-based deep convolutional neural network framework, which can execute CNN models for image recognition tasks. However, it only uses GPUs to accelerate the inference without considering other heterogeneous local processors on mobile platforms, such as CPUs, DSPs, and so on. DeepX [3] is also a framework to enable the execution of Deep Neural Networks (DNNs) and CNNs on mobile devices. It tries to use all local processors available even clouds, which can't avoid privacy leaks and may lead to lower energy-efficiency due to its greedy strategy.

Different from previous works, this paper concerns more about energy-efficiency of CNN-based mobile applications. An adaptive algorithm is proposed to compute the energy-efficiency of all local processors without requiring users to input device information. Based on the computed energy-efficiency, an energy-efficient device processor combination can be found to perform CNN inference in parallel. In summary, main contributions of this paper include:

- We propose a novel algorithm to adaptively compute the energy-efficiency of all local processors available on the target mobile platform and further obtain an energy-efficient combination to execute the CNN inference.
- We propose an easy method based on the above algorithm to split computation tasks across the selected energy-efficient device processor combination.
- We describe the implementation of our algorithms on ODROID-XU3. Experimental results show that our proposed strategy provides 3.67× higher energy-efficiency with only 9.7% performance degradation on average compared with the greedy strategy which tries to use all local processors available.

The rest of this paper is organized as follows: Section 2 provides necessary background about the development trends of mobile SoCs and the definition of Energy Delay Product (EDP), then analyzes the energy-efficiency of CNN inference executing on mobile CPU and GPU respectively and describes our motivation. Section 3 explains our proposed strategy in detail. The experiments and evaluation results are described in Section 4. Section 5 lists the related work. Finally, we conclude our work and plan the future works in Section 6.

0-7695-6329-5/17/31.00 ©2017 IEEE
DOI 10.1109/ISPA/IUCC.2017.00074

2. Backgrounds and Motivation

Recently, there are continuous innovations taking place in mobile architectures. Apart from high-performance multi-core CPUs, GPUs and DSPs have also been integrated into embedded SoCs, such as Snapdragon 820. Such advances give us a chance to execute CNN inference on local mobile devices. What's more, some specialized hardware like DianNao [5] and DaDianNao [6], which are very suitable to perform deep learning algorithms, may also be integrated into mobile devices in the future. With so many heterogeneous local device processors, we have to find an energy-efficient way to use them since the mobile phone battery capacity is limited.

An already well-known metric for evaluating energy-efficiency of a given program execution is Energy Delay Product (EDP) [7], which is a "smaller is better" metric. In our work, it is expressed as the product of energy consumption (E) and execution time (T) when performing a complete CNN inference, as shown in the following equation:

$$EDP = E \times T \tag{1}$$

Different from previous works in this area, which either failed to make good use of heterogeneous hardware resources (e.g., using only CPU or GPU) or tried to utilize all heterogeneous local device processors available, this paper mainly explores how to use heterogeneous computing to execute CNN inference locally in an energy-efficient way.

It is shown in Table 1, when using the mobile CPU to execute a complete CNN inference on ODROID-XU3 (see more details in Section 4), we get 11.94 s for average execution time and 52.12 J for average energy consumption even if we have used NEON SIMD instructions and launched enough threads. However, the average execution time and energy consumption can be reduced to 1.9 s and 1.48 J by using GPU to do the same work. It is true that the execution time will be less if we use both CPU and GPU to execute CNN inference in parallel, but the energy-efficiency will be very low. The reason can be seen from EDP values in Table 1, which shows that the energy-efficiency of GPU is about 221 times higher than that of CPU during the CNN inference. Because of the limited battery capacity, mobile systems must balance latency (delay) and battery (energy) usage for computation.

TABLE 1. COMPARISON BETWEEN CPU AND GPU DURING CNN INFERENCE

Processors	Time (s)	Energy (J)	Power (W)	EDP(J*s)
CPU	11.94	52.12	4.37	622.31
GPU	1.9	1.48	0.78	2.81

Based on the above analysis, we can conclude that it is necessary to find an energy-efficient combination of heterogeneous local processors to execute CNN inference instead of simply using all of the local processors on mobile platforms.

3. Proposed Strategy

Figure 1 shows the overview of the proposed strategy. Firstly, to find the energy-efficient device processor combination on a target mobile platform for performing CNN inference in parallel, the CNN-based application automatically measures all the energy-efficiency of different local processors based on the proposed algorithm. Then, once the desired device processor combination is found, we use an approach to divide computation tasks into different sizes according to the performance of each selected processor. Finally, all these tasks will be processed simultaneously by corresponding processors.

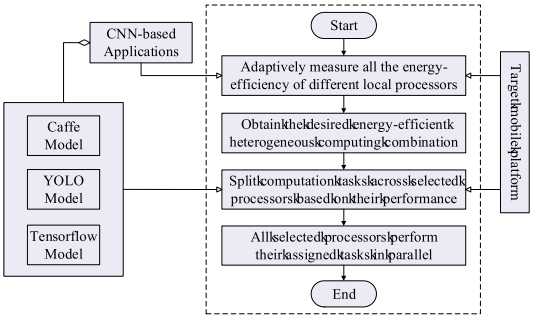


Figure 1. Overview of the proposed strategy

3.1. Search for Energy-Efficient Device Processor Combination

Algorithm 1 provides the pseudo-code of the search process. To perform inference on local device processors, a pre-trained CNN model with the structure and parameters is required. Since we only need its structure and weights, the model can be trained by Caffe, Tensorflow, YOLO [8], etc. Besides, the default value for the target energy-efficiency is set to 1, which can also be configured by users.

Since the desired energy-efficient heterogeneous computing combination must be a subset of the collection of all devices, all device processors available are collected into a container (line 1). With our approach, users have no need to measure the performance and power consumption in advance. On the contrary, the algorithm performs these measurements during the first several running processes and computes the energy-efficiency for each local device processor automatically (lines 2 - 7). These runtime measurements can cause some overhead, but it only happens on first several processes of running the application, so the overhead can be amortized over the whole use cycle of the application. After that, the processor with minimum execution time will be used as a reference (line 8), through which the performance and power consumption of each processor can be normalized (lines 9 - 12). Note that the relative performance of $processor_i$ should be obtained by $\frac{\frac{1}{t_i}}{\frac{1}{t_{min}}} = \frac{t_{min}}{t_i}$ (line 10), because the reciprocal of execution time is referred to as the performance of each processor.

To take full advantage of parallel processing, the amount of computation tasks assigned to each processor is configured dynamically by considering the processor performance. From line 10, we deduce that the relative performance of the reference processor (the processor with minimum execution time) is 1 and all others are less than 1. Hence, the task ratio

Algorithm 1 Search for energy-efficient device processor combination.

Input: (i) The CNN structure and pre-trained model parameters, including weight and bias matrices; (ii) Threshold EDP_{TH} for energy-efficiency (default value is 1).

Output: The desired energy-efficient heterogeneous computing combination S_{HC} and the relative performance of these selected processors $Perf$.

1: Exploring all the heterogeneous local device processors(e.g., GPUs, CPUs, DSPs and may DianNaos [5]), and put them in the array S_{HC};
2: **for all** $processor_i \in S_{HC}$ **do**
3: Running the complete CNN inference on $processor_i$;
4: Then its execution time t_i, energy consumption e_i and average power consumption p_i are stored in arrays T, E and P, respectively;
5: Computing its EDP_i and save it in the array EDP
6: ▷ using Equation 1;
7: **end for**;
8: Finding the processor with minimum execution time. Its execution time and average power consumption are referred to as t_{min} and p_{ref} respectively.
9: **for all** $(t_i \in T) \wedge (p_i \in P)$ **do**
10: Computing relative performance $perf_{ri} = \frac{t_{min}}{t_i}$ and add $perf_{ri}$ to $Perf$;
11: Computing relative power consumption $p_{ri} = \frac{p_i}{p_{ref}}$ and add p_{ri} to P_r;
12: **end for**;
13: **repeat**
14: Computing relative execution time $t_r = \frac{1}{\sum_{perf \in Perf} perf}$;
15: Computing total relative power consumption $p_{tr} = \sum_{pr \in P_r} pr$;
16: Computing relative EDP $edp_r = (p_{tr} * t_r) * t_r$;
17: **if** $edp_r \geq EDP_{TH}$ **then**
18: Finding the processor with maximum EDP value, and its index is i;
19: Removing $processor_i$ from S_{HC};
20: Removing $perf_{ri}$ from $Perf$;
21: Removing p_{ri} from P_r;
22: **end if**
23: **until** $(edp_r < EDP_{TH})$
24: **return** S_{HC} and $Perf$;

for the reference processor can be obtained by the following equation:

$$ratio_{ref} = \frac{1}{\sum_{perf \in Perf} perf} \quad (2)$$

It is assumed that the time for executing the complete CNN inference on the reference processor serially is t_s. Then the parallel-processing time will be $ratio_{ref} \times t_s$ (see Section 3.2 for further discussion). t_s can be simply viewed as 1 because of the normalization, so the relative parallel-processing time (t_r) is equal to $ratio_{ref}$. Since the relative execution time and power consumption have been computed, the EDP of these selected processors (edp_r) can be calculated from Equation 1 (lines 14 - 16). Then, the collection of selected processors will be updated by comparing edp_r with the target energy-efficiency (lines 17 - 22). The procedure described above is continually repeated until the desired energy-efficient heterogeneous computing combination is found. Finally, the collection of selected processors and their relative performance parameters will be returned.

Note that the energy-efficient device processor combination can be reused for the same application, so the search process above only needs to be done once.

3.2. Division of Computation Tasks

After the search step, computations will be split across the selected processors according to the results obtained by Algorithm 1. Because we want a minimal parallel-processing time, the waiting time among different processors should be minimized. Hence the key idea of the computation task division approach is that the task size assigned to a local processor is based on its performance.

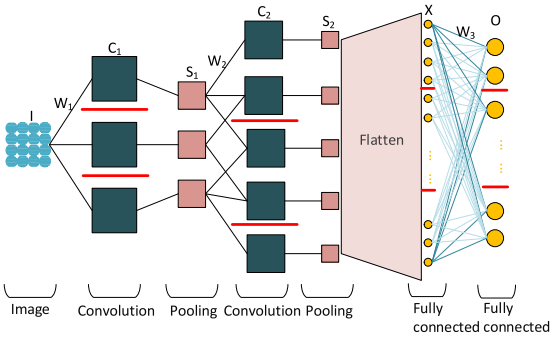
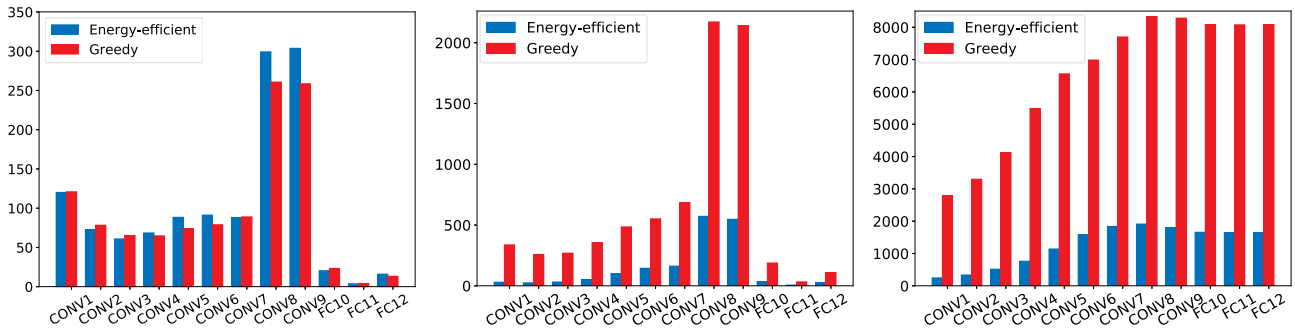


Figure 2. Diagram for the division of computation tasks, where each red line denotes a division point.

Given a pre-trained CNN model, we define a computation task as an output unit in each layer, i.e., the number of computation tasks equals to that of output units for each layer. Specifically, a computation task is a convolution operation for convolutional layers or an inner product operation for fully connected layers. Figure 2 illustrates an overview of the division of computation tasks, and each red line denotes a division point.

Algorithm 2 details the approach to computing the task ratio for each processor in the obtained energy-efficient device processor combination. As mentioned in Section 3.1, the amount of computation tasks assigned to each processor is decided by the processor performance. So the relative performance parameters obtained by Algorithm 1 are used to compute the task ratio for each selected processor (lines 1 - 4). With the task ratio array R for all selected processors, the amount of computation tasks can be calculated by the product of task ratio and the number of output units in each layer. As a result, high-performance processors are assigned with more computation tasks while low-performance processors are assigned fewer tasks. Then all different sizes of tasks can almost be finished simultaneously. Finally, the



(a) Execution time of each layer (ms) (b) Energy consumption of each layer (mJ) (c) Power consumption of each layer (mW)

Figure 3. Comparisons of execution time, power consumption, and energy consumption

system reaches a load balance and utilizes the computing resources of heterogeneous platforms in an energy-efficient way. The overhead of this step is negligible because it is only needed to perform once and its running results (task ratio array R) will be saved locally to be reused.

Algorithm 2 Computation task ratio for each processor in energy-efficient device processor combination.

Input: The relative performance of combination $Perf$.
Output: Task ratio R for each selected processor.

1: **for all** $perf_i \in Perf$ **do**
2: Computing the task ratio of $process_i$:

$$r = \frac{perf_{ri}}{\sum_{perf \in Perf} perf}$$

3: Adding r into the array R;
4: **end for**;
5: **return** R;

4. Experimental Evaluations

We perform experiments on the ODROID-XU3 development platform, which carries a Samsung Exynos5422 processor with ARM Mali-T628 MP6 GPU inside and runs an Android v5.1.1 with the Linux kernel v3.10.9. The Samsung Exynos5422 processor utilizes ARM's big.LITTLE technology and comprises ARM Cortex-A15 quad-core (big) and Cortex-A7 quad-core CPU (LITTLE). In experiments, we create 8 threads and use NEON SIMD instructions to perform computation tasks for the CNN inference running on CPU. The ARM Mali-T628 MP6 GPU has two GPU clusters, in detail, GPU cluster 0 and GPU cluster 1 are composed respectively of 4 and 2 ARM Mali-T628 cores. So all the local device processors on ODROID-XU3 are a CPU processor and two GPU devices. The power is measured by the integrated power analysis tool of ODROID-XU3. Namely 4 current/voltage sensors measure the power consumption of the big A15 cores, Little A7 cores, GPUs, and DRAMs individually.

Based on DeepSense [4], we implement the proposed strategy. And the implementation has been used to develop an intelligent surveillance system (Android Application), which can autonomously detect the object captured by a camera. The intelligence of this application is powered by the convolutional neural network, where Tiny YOLO [8] model is used. Note that Tiny YOLO has 9 convolutional layers and 3 fully connected layers.

According to Algorithm 1 and Algorithm 2, our strategy will get an energy-efficient device processor combination (containing two GPU device processors here), and we compare it with the greedy strategy which simply uses all local processors available (containing a CPU and two GPU device processors here). Figure 3(a) presents the execution time of 9 convolutional layers and 3 fully connected layers. From it, we can also find that the greedy strategy is just slightly better than ours in performance, and the gap is smaller in fully connected layers. The reason is that GPUs are better at parallel computing than CPUs even if CPUs use multi-cores and SIMD instructions. In Figure 3(a), it can also be seen that the execution time of some layers in the greedy strategy is longer than that in our method. This is because the greedy strategy has more communication overhead of task division, which plays a dominant role in the layers with fewer output nodes.

Figure 3(c) and Figure 3(b) illustrate the comparison of the energy and average power consumption for each layer. Different from Figure 3(a), they show that the energy and average power consumption of our strategy are significantly lower than those of the greedy strategy for all layers. This is due to the fact that mobile GPUs are usually ultra-low-power ones. For example, in Table 1, GPU's power consumption is only 0.78 W while CPU's is 4.37 W when using them to execute CNN inference respectively.

Figure 4 shows a comparison among different heterogeneous computing combinations in four aspects: average execution time, average energy consumption, average power consumption and average energy-efficiency (EDP) of a complete CNN inference. The *Single GPU* uses one GPU device processor to perform CNN inference while the *Energy-efficient* uses two GPU device processors (selected by our strategy) to execute CNN inference in parallel. The *Greedy* uses all local device processors available (containing a CPU and two GPU device processors here) to perform parallel CNN inference. The bar chart clearly illustrates that the average execution time reduces but the average energy and power consumption increase as the number of used device processors increases. For EDP values shown in Figure 4,

our strategy (Energy-efficient) achieves a better energy-efficiency, which is 3.67 × higher than that of the greedy strategy. Meanwhile, the performance of our strategy has only declined by 9.7%. The experiment proves that utilizing all local device processors available can achieve the highest performance but it can also lead to a much lower energy-efficiency.

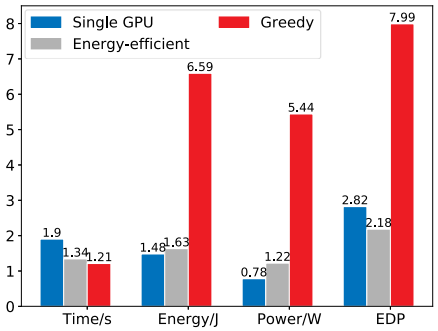


Figure 4. Comparison among different heterogeneous computing combinations

5. Related Work

There are lots of studies focusing on the optimization of running deep learning models on mobile devices. Accelerating the CNN inference by using matrix decomposition and approximation to compress each convolutional layer is a research hotspot now [9], [10]. DeepX [3] and DeepSense [4] used heterogeneous local computing to offload computation onto mobile GPUs. In addition, some methods on specialized hardware implementations of deep networks are also proposed [5], [6], [11], [12]. However, these studies do not consider the development trends of mobile SoCs or explore the energy-efficiency of heterogeneous computing when executing the CNN inference on mobile platforms, while this work does.

6. Conclusion and Future Work

Although there are not so many heterogeneous processors on mobile platforms now, there is an obvious trend integrating different heterogeneous processors into a mobile device to perform different tasks. As mentioned earlier, owing to the innovation of processor architecture, DianNao can process the neural network computation 117.87 × faster than a 128-bit 2GHz SIMD processor with the power consumption of 485 mW [5], which is even lower than that of mobile GPUs. So, this paper takes a big stride towards how to utilize mobile device processors to run deep-learning-based applications in an energy-efficient way. We have realized our proposed algorithms on ODROID-XU3, and the experimental evaluation shows that the proposed approach provides 3.67× higher energy-efficiency with only 9.7% performance degradation on average compared to the greedy strategy which tries to use all local processors available.

Future plan targets to take into account communication overhead of task division to achieve a higher energy-efficiency. Besides, a method for more fine-grained computation task division is also needed to be studied.

Acknowledgments

This work was supported by the National Science Foundation of China under grants (No.61379040, No.61772482, No.61202053, No.61222204, No.61221062). The authors deeply appreciate many reviewers for their insightful comments and suggestions. Jiangsu Provincial Natural Science Foundation (No.SBK201240198), Fundamental Research Funds for the Central Universities No. WK0110000034, and the Strategic Priority Research Program of CAS (No.XDA06010403).

References

[1] Keiji Yanai, Ryosuke Tanno, and Koichi Okamoto. Efficient mobile implementation of a cnn-based object recognition system. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 362–366. ACM, 2016.

[2] Seyyed Salar Latifi Oskouei, Hossein Golestani, Matin Hashemi, and Soheil Ghiasi. Cnndroid: Gpu-accelerated execution of trained deep convolutional neural networks on android. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 1201–1205. ACM, 2016.

[3] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Information Processing in Sensor Networks (IPSN), 2016 15th ACM/IEEE International Conference on*, pages 1–12. IEEE, 2016.

[4] Loc Nguyen Huynh, Rajesh Krishna Balan, and Youngki Lee. Deepsense: A gpu-based deep convolutional neural network framework on commodity mobile devices. In *Proceedings of the 2016 Workshop on Wearable Systems and Applications*, pages 25–30. ACM, 2016.

[5] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, volume 49, pages 269–284. ACM, 2014.

[6] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE Computer Society, 2014.

[7] Ricardo Gonzalez and Mark Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of solid-state circuits*, 31(9):1277–1284, 1996.

[8] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[9] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[10] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[11] Qi Yu, Chao Wang, Xiang Ma, Xi Li, and Xuehai Zhou. A deep learning prediction process accelerator based fpga. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 1159–1162. IEEE, 2015.

[12] Chao Wang, Lei Gong, Qi Yu, Xi Li, Yuan Xie, and Xuehai Zhou. Dlau: A scalable deep learning accelerator unit on fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(3):513–517, 2017.