Deep Learning Inference on Commodity Network Interface Cards

Giuseppe Siracusano NEC Labs Europe
Davide Sanvito Politecnico di Milano
Salvator Galea Univ. of Cambridge
Roberto Bifulco NEC Labs Europe

Abstract

Artificial neural networks' fully-connected layers require memory-bound operations on modern processors, which are therefore forced to stall their pipelines while waiting for memory loads. Computation batching improves on the issue, but it is largely inapplicable when dealing with time-sensitive serving workloads, which lowers the overall efficiency of the computing infrastructure. In this paper, we explore the opportunity to improve on the issue by offloading fully-connected layers processing to commodity Network Interface Cards. Our results show that current network cards can already process the fully-connected layers of binary neural networks, and thereby increase a machine's throughput and efficiency. Further preliminary tests show that, with a relatively small hardware design modification, a new generation of network cards could increase their fully-connected layers processing throughput by a factor of 10.

1 Introduction

The processing required by a large share of neural network (NN) serving workloads is memory-bound. For instance, [7] reports that over 60% of the Google's NN inference workload is due to Multi-layer Perceptrons (MLPs), which essentially are a sequence of fully-connected (FC) NN layers.

The memory-bound nature of this workload is evident in Fig. 1, which reports the Istruction per Cycle (IPC) of an Intel CPU when processing a three layers MLP with about 121M parameters. The hardware processing pipeline of that CPU is able to achieve an IPC above 3 when used efficiently, that is, it can perform three operations in a single clock cycle. Conversely, when processing the MLP's FC layers, it provides less than 0.5 IPC (cf. label "1" in Fig. 1), since the CPU is forced to wait for data loads. This is due to the relatively low arithmetic intensity of FC layers processing.

Batching several inputs together, and performing NN inference on them at once, allows for re-using a loaded model's parameters several times, thereby increasing the arithmetic intensity. However, increasing the batch size makes inference latency grow. For example, in Fig. 1, changing the batch size from 1 to 128 increases the IPC from 0.5 to 2.7, but it also increases the processing latency by almost 10x, from 40ms to about 400ms. For time-sensitive serving workloads, which may include the inference result as part of a user-facing service, such increase is usually not tolerable [19]. Therefore, the maximum batch size is limited to rather small values.

Another solution is model quantization, which reduces the precision of NN inputs and parameters, e.g., using 8bit values, thereby reducing memory requirements, while having little impact on model accuracy. Here, a promising direction is binary NNs [2], which use just one bit for both NN's inputs and activations, while achieving somewhat surprisingly good accuracy results. Table 1 shows it, reporting the accuracy of a binary MLP for an MNIST classification task, and a VGG16 [18] modified to replace the final 3 FC layers with binarized FC layers, for classification on the CIFAR10 dataset.

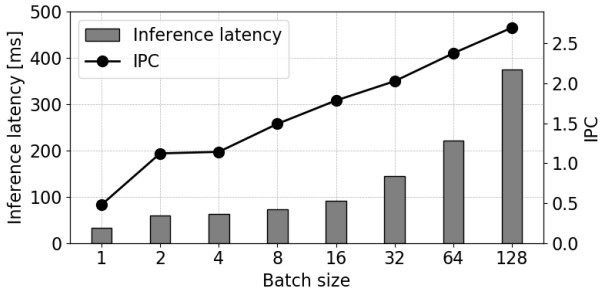

Figure 1: Latency-efficiency trade off.

Table 1: Regular vs binarized NNs

	MLP	Bin-MLP
Dataset	MNIST	
Layers	3 (bin)FC	
Acc.	98.7%	98.4%
	VGG	Bin-VGG
Dataset	CIFAR10	
Layers	13 conv, 3 (bin)FC	
Acc.	94.0%	94.0%

Interestingly, binarized models have also the effect of simplifying the arithmetic operations required to compute a NN, replacing matrix multiplications with bitwise operations, such as XOR. This enables the use of a machine's hardware component that were originally designed for different tasks.

In this paper, taking advantage of this observation, we consider the option of improving a NN serving system's efficiency and throughput using hardware deployed in a machine's networking subsystem, i.e., at the Network Interface Cards (NICs). More specifically, we believe that network packet processing and NN memory-bound inference workloads may have complementary traits, which makes processors developed for networking re-usable, to some extent, for the efficient processing of NN inference. In fact, high-end NICs [11, 22] can handle tens of Gigabits per second (Gbps) of data, they sit already on the data path of an inference request, and are designed to take advantage of the mostly parallel nature of the network traffic processing, which happens on a per-packet basis and nicely fits the per-neuron widely parallel processing model of NNs.

To verify our thesis, we implement – `toNIC`– a system to process binarized FC layers using a commodity SmartNIC. The experimental results confirm our intuition. Using a single commodity NIC, clocked at 800MHz, and without affecting the NIC's main packet forwarding function performance, we are able to improve a machine inference performance/power ratio by about 4%, even if such machine hosts a powerful Intel multi-core CPU clocked at 3.7GHz. In preliminary tests, we further explore the extension of an open source SmartNIC implementation, the NetFPGA [22], to evaluate the cost of introducing better support for NN inference. Our results show that, with a relatively small increase in the NIC's hardware resource requirements, the NN processing throughput performance could improve of a factor of 10 or more.

2 toNIC

This section presents the design and implementation of `toNIC`, an inference server that runs on SmartNICs and can serve inference workloads for binarized NNs composed of FC layers.

While being an integral part of the machines, the networking subsystem is generally transparent to the NN serving framework. However, it includes hardware components such as NICs, which have recently evolved into programmable devices. Programmable NICs include a processor, often specialized for the processing of network packets, a variable amount of DRAM, and one or more network ports. The processors can be of different types, including general purpose CPUs [10], specialized network processors [11] and even FPGAs [22]. In this work, we will focus on network processor-based SmartNICs. These can have price and power consumption comparable to traditional NICs. Furthermore, they are generally faster in handling network packets when compared to CPU-based SmartNICs, which makes them an increasingly common choice for several deployments [13]. In particular, we will use Netronome SmartNICs based on the NFP4000 processor [11].

NFP4000 primer. The Netronome NFP programmable architecture is shown in Fig. 3. Since network traffic is a mainly parallel workload, with packets belonging to independent network flows, NFP4000 devices are optimized to perform parallel computations, with several processing cores (micro-engines, MEs). Each ME has 8 threads, which share local registries that amount for a total of 4KB. MEs are further organized in islands, and each island has two shared SRAM memory areas of 64KB and 256KB, called CLS and CTM, respectively. Finally, the chip provides a memory area shared by all islands, the IMEM, of 4MB SRAM, and a memory subsystem that combines two 3MB SRAMs, used

as cache, with larger DRAMs, called EMEMs. MEs can communicate and synchronize with any other ME, irrespective of the location (i.e., same or different islands).

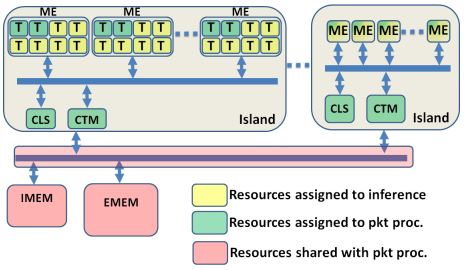

Figure 2: Inference and packet processing threads distribution.

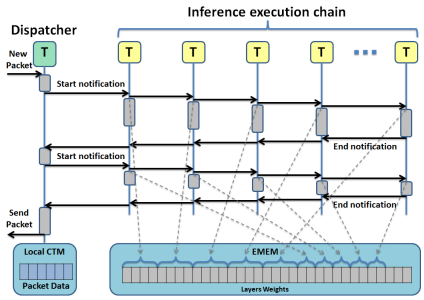

Figure 3: Inference threads notification and execution process.

Design. The main requirement of `toNIC` is to provide its inference function while preserving high-performance network communications. In our current design, we clearly separate the two functions, dedicating a configurable number of MEs' threads to the inference execution and the remaining ones to packet forwarding. Fig. 2 shows that we distribute the MEs' threads dedicated to inference across different islands. This is somewhat counter-intuitive, as one would expect that a co-location of processing elements working for the same task would yield better performance and more efficient synchronization. However, it should be noted that each island is provided with the fast CLS and CTM memories, which play an important role during packet processing. In fact, packet processing happens on a nanoseconds time scale, which implies a small time budget allocated to the processing of each packet. Instead, NN inference usually happens in a time budget of hundreds µs. Therefore, having access to fast local SRAMs is required to provide packet forwarding at low latency and high throughput, and assigning all the MEs of an island to inference would immediately make that island's fast memories unavailable to the packet processing task.

As we will see in Sec. 3, this design decision has the immediate effect of limiting the usefulness of batching computation for inference in `toNIC`. That is, weights cannot be cached in a local memory, being it dedicated to packet processing.

To combine forwarding and inference tasks, the packet forwarding threads work also as dispatchers for the incoming network packets that contain an inference request, e.g., encoded in an efficient remote-procedure-call (RPC) protocol. More in detail, at NIC's boot, *dispatching ME's threads* registers themselves for packet reception notifications. When a new network packet is received, the NFP4000 copies the packet content in the CTM memory of a dispatching thread's island, and generates a reception notification. At this point, the dispatching thread can parse the packet and invoke the forwarding function, if the packet is not destined to `toNIC`, or the inference function otherwise. The forwarding function is executed together with other non-programmable subsystems of the NFP4000, while the inference function is executed by the MEs' threads dedicated to inference.

Inference processing. A dispatching thread works as a coordinator for the execution of an inference, by starting the processing of one NN layer, and waiting for the result before starting the processing of the next layer. Fig. 3 depicts this process. The MEs' threads dedicated to inference are organized in a statically defined chain, with each ME knowing its predecessor and successor MEs' threads at boot time. To start processing a layer, the dispatching thread notifies the first thread in the chain with a *start notification*, which is then propagated throughout the chain.

After receiving the start notification, and sending it to the next thread in the chain, a thread performs the computation of a subset of the current layer's neurons, with the actual number depending on the number of neurons in the layer and the level of configured execution parallelism. Each of the thread is an executor from the perspective of `toNIC`, so the configured number of executors (threads) defines the level of parallelism.

For each of the neurons, an executor performs an XOR of the input with the weights vector, a population count operation and a comparison. The NN model input is stored in the packet data, located in the dispatcher thread's local CTM, which can be accessed also from other islands although paying the extra clock cycles required to do a cross-island read. The model's weights are stored in

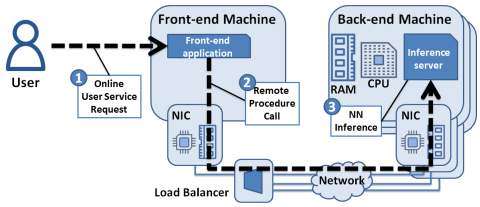

Figure 4: NN system overview.

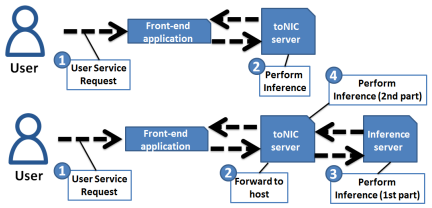

Figure 5: End to end system.

the DRAM-backed EMEM, in a contiguous memory space, which allows an executor to directly point to the weights of a set of neurons given its position in the execution chain. At the end of a layer computation, each executor writes its final result to the global IMEM memory, from which the dispatching thread can read it.

The last executor in the chain sends an *end notification* to its predecessor after writing its portion of the result to IMEM. The notification is propagated backward through the chain as all executors conclude their computations and write their results, until it is received by the dispatching thread. Here, either the computation for a new layer is started or the result is encapsulated in a network packet. The packet is then finally handled by the forwarding function.

End-to-end operations. In an end-to-end inference system [12, 3] (cf. Fig. 4), `toNIC` can work in two different configurations (cf. Fig 5). In stand-alone mode, the whole NN inference is performed on the NIC (e.g., for MLPs inference). In co-processor mode, the NN inference is performed in combination with a traditional NN serving framework (e.g., for convolutional NNs inference). That is, for cases in which FC layers are placed at the end of a NN, like in some convolutional NNs, the compute-intense convolutional layers can be processed by the inference server on the machine, while the memory-intense FC layers are offloaded to `toNIC`.

3 Evaluation

We evaluate `toNIC` components and the system as a whole measuring key performance indicators such as throughput, latency and power consumption. We compare our prototype with a serving system for binary layers optimized for the Haswell CPU – `bnn-exec`– which we developed specifically for this evaluation, to leverage Haswell's AVX instructions[1]. Both execute a single FC layer (including activations) with 4096 binary inputs and a variable number of neurons, from 2k to 16k. When doing latency measurements, we use a single core of the Haswell, while we consider all 4 cores for throughput benchmarks. Instead, `toNIC` always runs using 256 executors (6 threads from each of 42 MEs, plus 4 threads from another ME).

Latency. Our first test measures the per-inference latency, when processing a single FC layer (Fig. 6). For layers between 2k and 16k neurons (8M to 67M weights), `toNIC` achieves a processing latency which is only 4 times higher than `bnn-exec`'s one, varying between 400µs and 2700µs. Considering that the Haswell CPU has a clock frequency more than 4 times higher than NFP's 800MHz, i.e., each operation is effectively executed in less than a fourth of the time, this shows that the NFP is slightly more efficient than the Haswell in performing the FC layer operations. Even considering the larger FC layer with 67M weights, this processing latency should be acceptable for most applications.

Throughput. `toNIC` cannot perform batching, since it has to save hardware resources for packet forwarding tasks. In contrast, `bnn-exec` can batch executions, with the batch size being limited by the maximum allowed per-inference latency. We set this limit arbitrarily to 7ms, referring to [7]. This latency constraint allows `bnn-exec` to run with a batch size of 64, 32, 16 and 8 for the 2k, 4k, 8k and 16k neurons layers, respectively. Fig. 7 reports the results in terms of *FC layers per second (FC/s)*. For `bnn-exec` we plot the result obtained for each FC size with the above mentioned batch sizes (and running on the CPU's 4 cores). `toNIC`, despite unable to perform batching, and using only a subset of the NFP4000 resources, can still provide 4-5% of the `bnn-exec` throughput.

[1]Unless differently stated, we run all the tests on a machine equipped with an Intel Haswell E5-1630 v3 CPU and a single Netronome Agilio CX SmartNIC, with an NFP4000 processors and two 40Gbps ports. The Haswell is clocked at 3.7GHz, while the NFP at 800MHz. `toNIC` runs on the NFP, while `bnn-exec` runs on the CPU.

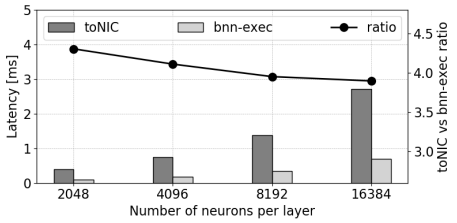

Figure 6: The processing latency of an FC layer.

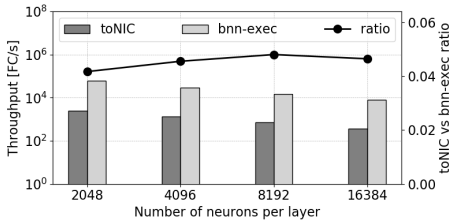

Figure 7: Throughput for different layer size.

Table 2: Power measurements when computing FC layers on NFP and CPU

	POWER (VS IDLE)	FC/S	FC/W VS IDLE
IDLE	69.4W	-	-
`BNN-EXEC`	145.9W (+75.5)	29520	386
`TONIC`	70.2W (+0.8)	1344	1680

Forwarding performance. While running the above throughput test, we load the NFP4000 with 80Gbps of network traffic to be forwarded (line rate of the NIC). For all packet sizes of 512B and bigger, the SmartNIC could forward traffic at line rate, i.e., both the packet forwarding function and the inference function could provide their maximum throughput while working in parallel.

Power measurements. The previous results tell relatively little about the actual performance of the evaluated systems, which is better captured by a metric related to power efficiency. In particular, we use the *FC layers per Watt (FC/W)* as an indicator of power efficiency, and as a proxy for the cost of running the system. The power consumption of our test machine when completely idle is 69.4W. For measuring the FC/W rate, we serially execute FC layers with 4k neurons (16M weights, 2MB in memory) on each processor, separately. We select this workload being a middle-ground between the best and worse performance of `toNIC` compared to `bnn-exec` (cf. Fig. 7). Furthermore, this allows both processors to minimize loads from DRAMs, since both of them have large enough SRAM caches to hold the 2MB of weights (EMEM's cache in the case of NFP, and L3 for Haswell). `toNIC` achieves a throughput of about 1.35k FC/s, with the overall machine power consumption rising to 70.2W, i.e., it takes 0.8W additional W of power to compute 1.35k FC/s. This yields a very promising 1.7k FC/W relative to the idle power consumption. When the Haswell CPU is used for executing FC layers, instead, the throughput is 29.5k FC/s, but the power consumption raises to 145.9W, with a performance of just 386 FC/W relative to the idle power consumption. That is, the NFP4000 yields a 4.3x better performance/power ratio (cf. Table 2).

To put this in perspective, we need to take into account the overall power consumption of the machine, and factor in the relative contribution given by `toNIC`. I.e., `toNIC` provides a 5% maximum throughput increment in terms of FC/s to the machine, in exchange of a 0.5% increase in power consumption, thereby increasing the overall machine FC/W by about 3.9%.

End-to-end test. In a final test we measure the end-to-end processing latency for a full inference task, using the MLP example of Table 1. We measure the latency from a second server connected back-to-back with our machine running `toNIC` (`bnn-exec`), measuring the time difference between an inference response and its request as seen by the second server. The latency with `toNIC` is 1.05ms, while with `bnn-exec` it is 0,83ms, when both devices have a *cold* cache. Here, it is interesting to notice that being `toNIC` running in the NIC, of the 1.05ms only 35µs are spent in network packet's reception and transmission, while for `bnn-exec` it takes 190µs only to handle the network traffic. Notice that these value include the overhead introduced by the measurement server. The additional latency of `bnn-exec` is due to the need to traverse the NIC, the PCIe bus and the host's machine network stack before `bnn-exec` can actually receive the input to perform inference.

4 Related Work

The closest related work is perhaps Microsoft's BrainWave [5], which uses network-attached FPGAs as accelerators for NNs. Instead, `toNIC` relies on commodity NICs, and explores opportunities to integrate NN inference processing capabilities in such devices. In this sense, our work could be

positioned in the larger trend of in-network computation. In fact, commodity programmable network devices are a relatively recent evolution [1], but are already becoming a commodity given the issues in further scaling general purpose CPUs performance. Since their introduction, researchers have been exploring ways to apply them to domains not necessarily related to networking, such as key-value store servers [9], distributed consensus [4] and more recently the acceleration of parameter servers for machine learning model training [17]. With these efforts we share only some common constraints, and in particular the need to efficiently encode information in network packets. In our previous work we showed that it is possible to run NN models using both switches [20] and NICs [16]. We follow and extend such works providing for the first time, to the best of our knowledge, a serving system design for commodity SmartNICs. Finally, model quantization and binary NNs [6, 8, 2, 15] closely relate to the applicability of `toNIC`.

5 Discussion and next steps

This paper positively answers to the question *can a machine's networking subsystem assist with memory-bound NN operations?*, and by doing so, it uncovers interesting emerging synergies among two so far independent research fields dealing with specialized hardware for deep learning and network devices. In fact, our `toNIC` prototype builds on the NFP4000, which does not have instructions that can efficiently deal with binary NNs, significantly limiting the achieved throughput.

What is the cost of modifying a SmartNIC to significantly *improve its inference throughput?*

We extend the design of an open source FPGA-based network card, the NetFPGA-SUME [22][2], in order to evaluate the cost of adding specialized circuitry for binary NN inference.

In the FPGAs domain, the cost of a design can be evaluated in terms of memory and logic (LUTs) resources, thus we extend the NetFPGA's reference switch design to introduce circuitry to handle binary NN, and evalutate the introduced overhead. In particular, we introduce a binary FC layer executor that can run 256 neurons in parallel, with up to 4096 input values, which requires memory to hold at least 1M weights, i.e., 131KB. The design is clocked at 200MHz, and can execute the 256 neurons with 4096 inputs in 16 clock cycles, i.e., approximately in 80 nanoseconds. We further extend the design to hold weights for 4096 neurons (a total of 16M weights), which requires 2MB of memory, in order to provide support for computation batching for an entire layer. Of course, this would require dedicating 2MB of expensive FPGA's block RAM to the FC layer executor, but would enable the execution of the entire layer to be performed in just 1.3µs. That is, the layer execution would be performed in 16 chunks of 256 neurons, each taking 16 clock cycles.

In a purely sequential execution this could already, theoretically, yield a throughput of 781k FC/s. Here, notice that this is somewhat expected, since binary NNs are well known for their ease and efficiency of implementation in hardware [21].

Admittedly, our implementation is relatively naive, handling just the case of FC layers with weights stored in the FPGA's block RAM, however, it provides some interesting insights on the possible cost of such hardware component. Our FC layer executor needs at most 2MB of block RAM, which can be either shared or dedicated, e.g., like the on-chip caches of the NFP4000. If block RAM is an issue, one can trade-off memory space for performance, lowering the requirement to just 131KB and moving the remaining weights to other types of memory (e.g., DRAMS). In terms of additional logic resources, i.e., LUTs, instead, our design needs only 679 LUTs. For reference, that is less than 1% of the logic required to implement basic SmartNICs operations. E.g., [14] needs 71k LUTs.

Concluding remarks. Overall, these preliminary numbers suggest that a solution that combines hardware for deep learning and network packet processing is viable and worth exploring. Our prototype, `toNIC`, can increase the tested machine's throughput by 5% with just 0.5% of additional power consumption, using already deployed commodity NICs. Extending an FPGA-based NIC prototype, we show that introducing a fully-connected layers executor can potentially increase the NIC's inference throughput by 10-100x, in the considered cases, while requiring relatively little additional hardware resources. Given these results, we believe that an interesting research opportunity opens at the confluence of the traditionally independent efforts in designing accelerators for neural networks and network packet processing devices.

[2]The NetFPGA-SUME platform is an x8 Gen3 PCIe adapter card containing a Xilinx Virtex-7 690T FPGA and four SFP+ ports providing four 10G Ethernet links.

References

[1] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 99–110, New York, NY, USA, 2013. ACM.

[2] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[3] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. Clipper: A low-latency online prediction serving system. In *NSDI*, pages 613–627, 2017.

[4] Huynh Tu Dang, Marco Canini, Fernando Pedone, and Robert Soulé. Paxos made switch-y. *SIGCOMM Comput. Commun. Rev.*, 46(2):18–24, May 2016.

[5] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. A configurable cloud-scale dnn processor for real-time ai. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 1–14. IEEE Press, 2018.

[6] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 4114–4122, USA, 2016. Curran Associates Inc.

[7] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12. ACM, 2017.

[8] Minje Kim and Paris Smaragdis. Bitwise neural networks. *CoRR*, abs/1601.06071, 2016.

[9] Xiaozhou Li, Raghav Sethi, Michael Kaminsky, David G. Andersen, and Michael J. Freedman. Be fast, cheap and in control with switchkv. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 31–44, Santa Clara, CA, 2016. USENIX Association.

[10] Mellanox. Mellanox, 2018. `http://www.mellanox.com/related-docs/prod_adapter_cards/PB_BlueField_Smart_NIC.pdf`.

[11] Netronome. Netronome AgilioTM CX 2x40GbE intelligent server adapter, 2018. `https://www.netronome.com/media/redactor_files/PB_Agilio_CX_2x40GbE.pdf`.

[12] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139*, 2017.

[13] OpenComputeProject. SPEC, MEZZ, OCS, Netronome NIC retrieved sept. 2018, 2018.

[14] Salvatore Pontarelli, Roberto Bifulco, Marco Bonola, Carmelo Cascone, Marco Spaziani, Valerio Bruschi, Davide Sanvito, Giuseppe Siracusano, Antonio Capone, Michio Honda, Felipe Huici, and Giuseppe Bianchi. Flowblaze: Stateful packet processing in hardware. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, 2019.

[15] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016.

[16] Davide Sanvito, Giuseppe Siracusano, and Roberto Bifulco. Can the network be the ai accelerator? In *Proceedings of the 2018 Morning Workshop on In-Network Computing*, NetCompute '18, pages 20–25, New York, NY, USA, 2018. ACM.

[17] Amedeo Sapio, Ibrahim Abdelaziz, Abdulla Aldilaijan, Marco Canini, and Panos Kalnis. In-network computation is a dumb idea whose time has come. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks, Palo Alto, CA, USA, HotNets 2017, November 30 - December 01, 2017*, pages 150–156, 2017.

[18] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[19] Ankit Singla, Balakrishnan Chandrasekaran, P Godfrey, and Bruce Maggs. The internet at the speed of light. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, page 1. ACM, 2014.

[20] Giuseppe Siracusano and Roberto Bifulco. In-network neural networks. *arXiv preprint arXiv:1801.05731*, 2018.

[21] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, pages 65–74, New York, NY, USA, 2017. ACM.

[22] Noa Zilberman, Yury Audzevich, G. Adam Covington, and Andrew W. Moore. NetFPGA SUME: Toward 100 Gbps as research commodity. *IEEE Micro*, 34(5), 2014.