Understanding the Power Consumption of Executing Deep Neural Networks on a Distributed Robot System

Ramyad Hadidi[1], Jiashen Cao[1], Matthew Merck[1], Arthur Siqueira[1], Qiusen Huang[1], Abhijeet Saraha[1], Chunjun Jia[1], Bingyao Wang[1], Dongsuk Lim[1], Lixing Liu[1], and Hyesoon Kim[1]

***Abstract*— Robots have access to an abundance of raw data. They need to understand their environment to perform their tasks and act upon local triggers. With the advancement of deep neural networks (DNNs), robots, with the help of powerful computing units, can now understand complex visual features, such as humans and objects. Since DNNs have high computation and power demands, to execute DNNs and achieve high performance, a robot requires state-of-the-art or domain-specific hardware. Although integrating such hardware in several scenarios is economically and logically sound (such as integrating GPUs in self-driving vehicles), for a handful of other scenarios several constraints limit this integration. For instance, unmanned aerial vehicles (UAVs) have a constraint on their weight which limit their ability to carry high-capacity batteries. Thus, we need a good understating of DNNs power consumption on robots and UAVs. To understand the power consumption of robots and UAVs that execute DNNs beside their operation, we study a two-robot system with limited computing capabilities. We utilize a framework that enables collaboration among several low-power devices for enabling real-time execution of DNNs. Our execution platform includes two iRobots, each of which has been equipped with an additional Raspberry Pi 3, the power source of which is the iRobot's battery. By executing an example DNN on interconnected Raspberry Pi 3s, while iRobots are operating, we quantify and study the power consumption effects of DNNs in our system.**

***Index Terms*— Deep Learning in Robotics and Automation, Distributed Robot System, Power Usage**

I. Introduction

Robots rely on raw data that is derived from their environment and act upon it. They need to process this raw information to extract useful data. With the emergence of deep neural networks (DNNs) and popularity of machine learning, we have been extending our capabilities to solve traditionally challenging problems such as computer vision, natural language processing, neural machine translation, and video recognition [1]–[3]. These new capabilities are extremely beneficial for robots because robots are usually at the forefront of the information that only could be comprehended by DNNs. While DNNs can aid robots tremendously, executing DNNs requires intensive computation power and access to a reliable source of energy. Therefore, in several use-cases, robots are enhanced with domain-specific or high-performance hardware to process DDNs. However, integrating such costly hardware leads to more expensive robots. (sometimes in multiple factors of the initial cost; for example, to add a high-performance GPU, which costs hundreds of dollars.) The co-design of the robot hardware for integrating the high-performance components usually incurs the high cost of redesigning and thus creates a barrier to execute DNNs directly on the robots. In addition, the weight constraints of several robots, such as unmanned aerial vehicles (UAV), limits their ability to carry large batteries or high-performance GPUs. Therefore, although executing DNNs on robots are greatly beneficial, a barrier caused by manufacturing cost and the various constraints of some robots/UAVs limits DNNs applicability.

In this short paper, to overcome the barrier of executing DNNs on inexpensive or constrained robots/UAVs, we study a scenario in which multiple low-power robots exist in an environment. In this scenario, DNNs cannot be executed on a single robot because of their high computational resource and power demand. However, ideally, by sharing the computational power of all the robots in the environment, we can move one step closer to the execution of DNNs. There have been several works to enable the execution of DNNs on low-power robots, embedded devices, or Internet of things (IoT) devices [6]–[13], such as collaborative computation between edge devices and the cloud [14]–[16], or customized mobile implementations [17]–[24]. For this work, we use Hadidi et al. [11], [13] framework to collaboratively execute AlexNet [25], an image-recognition DNN model, on two iRobots [4]. As shown in Figure 1, each iRobot is equipped with an additional Raspberry Pi 3 [5] that is connected the iRobot's battery through a voltage converter. To understand the power consumption of executing DNNs on this distributed robot system, we study the power consumption of both iRobot and Raspberry Pi 3 in four cases: idle (stationary robot), iRobot moving with no computation, DNN execution while iRobot is stationary, and DNN execution

Fig. 1: Two iRobot2 [4], equipped with two RPi3s [5].

[1]Computer Science School and Electrical Engineering Department, Georgia Institute of Technology, GA 30332, USA {rhadidi,jcao62,merckmatthew,arthurs,qhuang79, asaraha3,cjia98,bennywang,dlim46,lliu374} @gatech.edu, hyesoon@cc.gatech.edu.

This work is supported by NSF CSR 1815047.

while iRobot is moving. Since battery size is a constraining factor in several robots and UAVs, by understanding the power consumption of our system in several cases, we hope to shed more light on the details of executing DNNs in robots and UAVs.

II. Distribution and System

A. Distributing DNNs

In this paper, we use Hadidi et al. [11], [13] framework that enables an efficient, local, and distributed computation of DNNs close to the edge by using several resource-constrained devices or robots. A single low-power robot alone cannot handle all computations of DNNs [11]. Although with some optimizations, such as weight pruning [26], [27] and precision reduction [28]–[30], we can run limited versions of the current models on robots, Hadidi et al. solution is orthogonal to such techniques. Therefore, these techniques can also be applied on top of the mentioned solution. Since only one request exists in our environment, to distribute the computation of the DNN, we need to use model parallelism methods. Model parallelism is splitting the computation of a DNN across multiple robots, whereas data parallelism is processing the independent data concurrently. Model parallelism methods for different DNN layers are discussed in [12] extensively.

In summary, Hadidi et al. framework, either first profiles all the combinations [11] or uses real-time monitoring tools [12] to find an optimal distribution of a DNN model. Note that these methods are tailored toward single batch processing since we have limited number of request in our target environment (e.g., edge computation domains, UAVs, IoTs). The methods study the memory and compute footprints for each layer of a DNN model, and create an optimal work distribution for the DNN model. The target performance of the model is to reduce the latency of the model. Therefore, as discussed in [12], the methods extensively use several model-parallelism methods for fully-connected and convolution layers. Next we describe the utilized DNN model. We choose AlexNet because it is well-known. MobileNet models [19] are also a good candidate for our systems. However, because of their complex structure and large number of layers, we were unable to find an optimal distribution at the time of this submission. In fact, DNN choice is arbitrary in our implementation since by using [12] method, we are able to deploy any DNN model on our system. Therefore, the optimizations for MobileNet models will help us speed up the execution even more and are orthogonal to our work distribution.

B. AlexNet DNN Model

In the 2012 ImageNet large-scale visual recognition challenge (ILSVRC), a challenge for image recognition task, AlexNet [25] significantly outperformed all the prior competitors and won the challenge with a deeper convolution neural network (CNN) and more filters per layer. Figure 2 illustrates the model of a single-stream AlexNet, which consists of five convolution layers and three fully-connected layers. The model that we use has around 40M parameters (single-stream AlexNet).

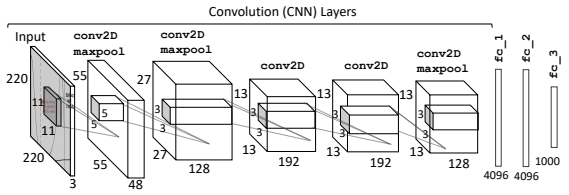


Fig. 2: AlexNet image-recognition model [25].

C. System Setup

We use two iRobot Roomba 600 as our robots, each of which are equipped with one Raspberry Pi 3. The power source of Raspberry Pi 3 is derived from iRobot's battery with a voltage converter. To control iRobot, we use iRobot Create 2 Open Interface [31] to send serial commands from the Raspberry Pi on top of it that is connected to iRobot's serial port. As our computation engines for executing DNNs on each iRobot, we utilize Raspberry Pi 3s, the specification of which is in Table I. On each Raspberry Pi, with the Ubuntu 16.04 operating system, we use Keras 2.0 [32] with the TensorFlow 1.0 [33] backend. To measure the power consumption of a single Raspberry Pi, as shown in Figure 1, we use a USB digital multimeter that logs measurements in an excel file (every 1s). To measure the power consumption of one robot, we use iRobot Open Interface to poll iRobot's battery voltage and amperage (every 100 ms) while performing our experiments. All experiments are done for approximately 3 minutes, including some idle time to show the baseline.

Currently, Raspberry Pis doe not share the output from DNN models with iRobots for path control. The current system is built as a showcase to show collaboration between robots. We plan to extend this system with a camera per robot, so that Raspberry Pis can control iRobots using the DNN model output. Our assumption in evaluation our system was that each robot records it own images and for processing uses collaboration. Therefore, each robot make its own decisions. We plant to extend this decision making to a collaborative decision since all the robots in an environment partially share same prospective. Similarly, in case of a connection failure, the data is loss. To solve this challenge, we can utilize coded distribution [34].

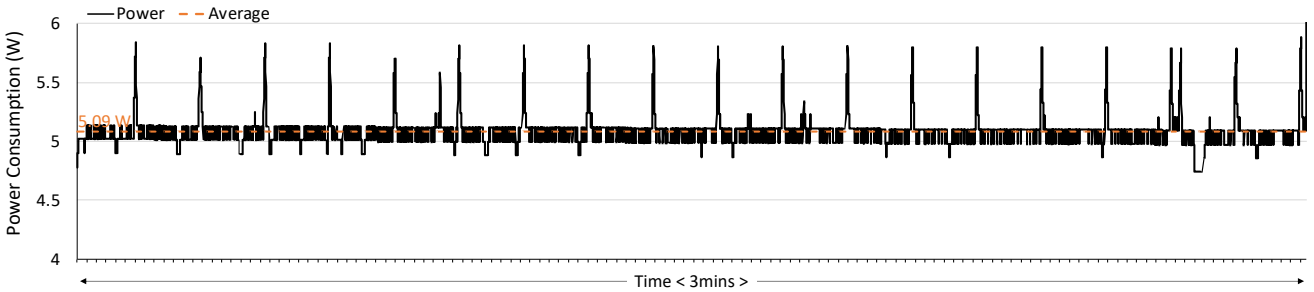
TABLE I: The specification of Raspberry Pi 3 [5].

CPU	1.2 GHz Quad Core ARM Cortex-A53
Memory	900 MHz 1 GB RAM LPDDR2
GPU	No GPGPU Capability

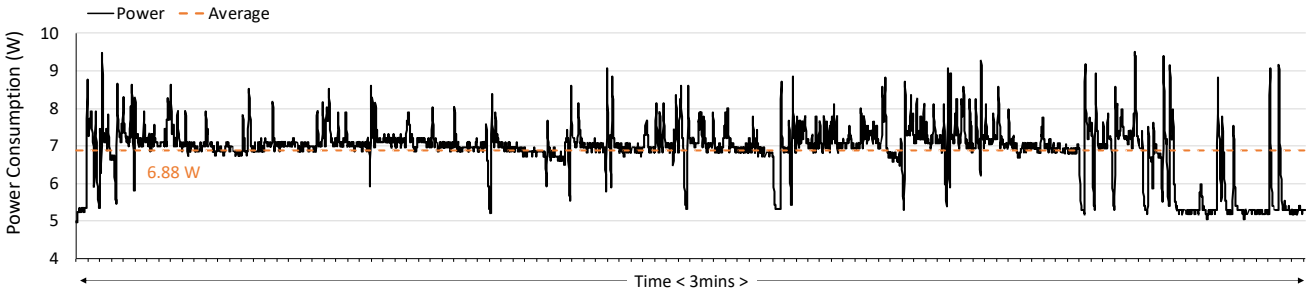
III. Experimental Results

A. Raspberry Pi Power Consumption

For understanding how DNN distribution affects the power consumption of the Raspberry Pi, in our first set of experiments, we measure the power usage of Raspberry Pi. In one experiment, as shown in Figure 4, we measure the



(a)



(b)

Fig. 3: Power consumption of iRobot in (a) idle mode (stationary) and (b) in movement. Power consumption includes all operations, motors, computer, and Raspberry Pi.

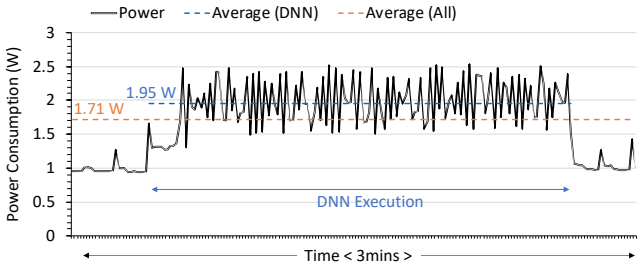


Fig. 4: Power consumption of a single Raspberry Pi 3 executing whole AlexNet.

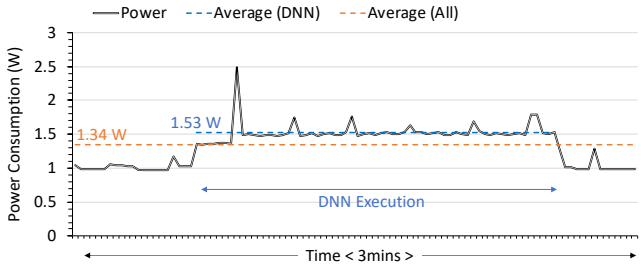


Fig. 5: Power consumption of a single Raspberry Pi 3 executing AlexNet in a distributed manner (total of two Raspberry Pis).

power consumption of a Raspberry Pi while it is executing the whole AlexNet model. As seen, the average power consumption is 1.95 W during DNN execution. We have also included some idle time in the figure to depict the change. The performance of one Raspberry Pi for AlexNet is 1.25 inferences per second. Therefore, around 200 inferences are done during our experiment. As a comparison, we also execute AlexNet on two Raspberry Pis and measure the power usage of one of them. The execution is performed by dividing the AlexNet model and executing each half on one Raspberry Pi. Figure 5 illustrates the trend in power usage in this experiment. As seen, although the idle power consumption of the Raspberry Pi is the same as the previous experiment, the power consumption during DNN execution is 1.53 W, less than that of the previous experiment. With two Raspberry Pis, our performance is around 3 inference per second, however, the power consumption of each single device is less than the case in which one device performs all the computations. This is because, with distribution, (i) less computation are performed per device, (ii) fewer memory operations are performed per device, (iii) each device has some idle time as a result of communication latency. Although as a whole system, the power consumption of the distributed system has been increased (i.e., power consumption one device with 1.95 W versus two devices each with 1.53 W, a total of 3.0 W), each individual device consumes less power.

B. iRobot and Raspberry Pi Power Consumption

In this section, we present the results of both iRobot and Raspberry Pi power consumption in several common cases. As discussed, we equipped each iRobot with a Raspberry Pi. We measure the power that is drawn from the iRobot's battery, which includes both power usage for powering up the Raspberry Pi and iRobot's movements. The first case is when iRobot is in idle mode (no movement, or stationary) with no computation. Figure 3a illustrates the power usage behavior in this case, which has an average power consumption of 5.09 W. The frequent spikes (around 0.8 W) in the figure is because of iRobot's frequent system checks. To see how physical activity affects power usage,

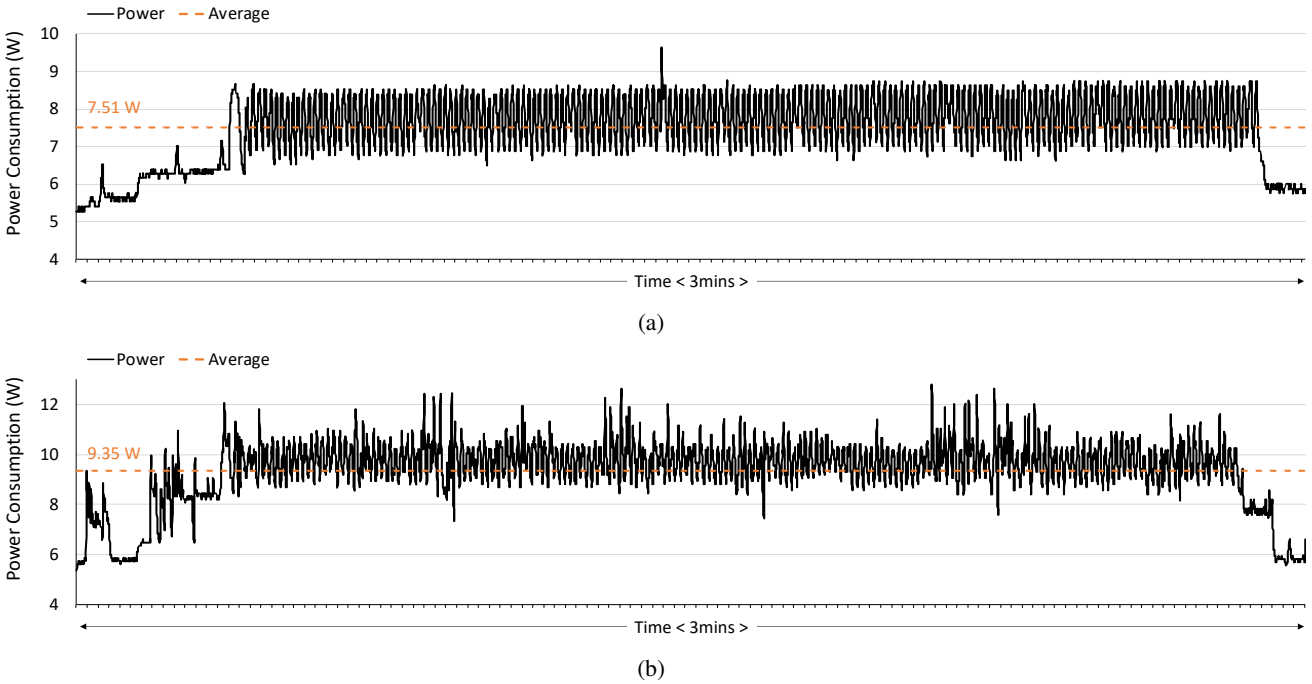


Fig. 6: Power consumption of iRobot in (a) idle mode (stationary) while executing AlexNet in distributed manner and (b) in movement while executing AlexNet in distributed manner. Power consumption includes all operations, motors, computer, and Raspberry Pi.

in Figure 3b, we measure the power consumption of iRobot while moving. The movement profile is random and depends on the environment and only the motors for movement are activated. As seen, the average power consumption is 6.88 W, around 1.8 W higher than the idle case. In addition, there are spikes as large as 3 W in the graph. In comparison with the idle case, in movement, the trend in power consumption is less predictable and contains large spikes.

To measure the effects on executing DNNs on robots, we execute AlexNet on the two Raspberry Pis in our system. Figure 6a illustrates the power consumption of one iRobot in stationary mode, while it is performing the computation of AlexNet (in a collaborative way with other iRobot). The average power consumption, in this case, is 7.51 W, and we have spikes around 2 W. Although the profile of power consumption should be similar to what we saw in Figure 5, since iRobot is not moving, the spikes are much larger than what we observed. We believe this is because of the unreliability of iRobot's battery in its ability to sustain a constant current to the Raspberry Pi or our circuitry in converting the voltage. In summary, DNN execution increases the power consumption from 6.88 W to 7.51 W, around 6% increase. Note that this increase only accounts for the dynamic power consumption for DNN execution. In fact, the addition of a Raspberry Pi increases the static power consumption of the system around 41% (Derived from 3.5 W average idle power consumption of iRobot with no Raspberry Pi, and 1.5 W average power consumption of Raspberry Pi).

Figure 6b depicts the power consumption of one iRobot in movement and while it is performing the computation for AlexNet (in a collaborative way). This case is the closest case to a real-world setting, in which the robot is acting on the previous outcome of the DNN, while the DNN is performing new computations. As seen, the average power consumption is 9.35 W, an 85% increase compared to the idle case with no computations. In some cases, the spikes are as large as 4.5 W. Also, executing the DNN has caused the power consumption to vary more frequently compared to the previous case with movement but with no DNN computation (Figure 3b). Such high spikes might limit Raspberry Pi capability in attaining a high performance since low power delivery may lead to different power saving settings in its CPU. To see the power consumption trends in various cases, Table II summarizes our results. Note that all the results include the static power consumption of both the iRobot and Raspberry Pi. As seen, DNN execution increases power consumption around 2.5 W (50% of the system in idle) and significantly increases the strength of spikes.

To extrapolate performance gain and energy trends of distributed computations on Raspberry Pis, we measure the performance and energy consumption of two, four, and six Raspberry Pis for AlexNet execution. Figure 7 depicts the performance and energy numbers of Raspberry Pi systems with two, four, and six number of interconnected devices. As seen with more Raspberry Pis, we achieve better per-

TABLE II: iRobot average power consumptions.

	Case	Average Power Consumption (W)	Spike Strength (W)
~~DNN~~	Idle	5.1	0.8
	Movement	6.9	3.0
DNN	Idle	7.5	2
	Movement	9.4	4.5

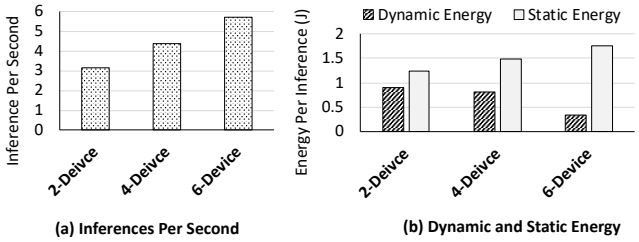


Fig. 7: The inferences per second (a) and power consumption (b) of various systems.

formance. A decreasing trend is also observed in dynamic energy consumption, similar to Figures 4 and 5. However, static energy consumption increases with the number of devices. This is because each Raspberry Pi comes with several components that are not necessary for us, and they consume extra energy.

IV. Conclusion

In this short paper, we studied the effects of executing DNNs in a distributed manner on the power consumption. The computation of our DNN, AlexNet, is distributed on Raspberry Pis that are mounted on iRobots and utilize its battery for their power source. We measured that executing DNNs increases the power consumption of the system around 2.5 W or 50%, while increasing the spikes strengths in power consumption. Moreover, we found out that increased number of devices leads to less power consumption in a system, which is beneficial for robots that have limited energy sources or have limits on their total weight such as UAVs.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[3] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[4] iRobot Inc., "irobot create 2 programmable robot," www.irobot.com/about-irobot/stem/create-2, 2019, [Online; accessed 15/03/19].

[5] R. P. Foundation, "Raspberry pi 3," www. raspberrypi.org/products /raspberry-pi-3-model-b/, 2017, [Online; accessed 15/03/19].

[6] Y. Wang, H. Li, and X. Li, "Re-architecting the on-chip memory sub-system of machine-learning accelerator for embedded devices," in *ICCAD'16*, 2016, pp. 1–6.

[7] Y.-D. Kim, E. Park, S. Yoo, *et al.*, "Compression of deep convolutional neural networks for fast and low power mobile applications," in *ICLR'16*. ACM, 2016.

[8] B. McDanel, S. Teerapittayanon, and H. Kung, "Embedded binarized neural networks," in *EWSN'17*, 2017, pp. 168–173.

[9] S. Bang, J. Wang, Z. Li, *et al.*, "14.7 a 288μw programmable deep-learning processor with 270kb on-chip weight storage using non-uniform memory hierarchy for mobile intelligence," in *ISSCC'17*. IEEE, 2017, pp. 250–251.

[10] R. LiKamWa, Y. Hou, J. Gao, *et al.*, "Redeye: Analog convnet image sensor architecture for continuous mobile vision," in *ISCA'16*. ACM, 2016, pp. 255–266.

[11] R. Hadidi, J. Cao, M. S. Ryoo, *et al.*, "Distributed perception by collaborative robots," *IEEE Robotics and Automation Letters (RA-L), Invited to IEEE/RSJ International Conference on Intelligent Robots and Systems 2018 (IROS)*, vol. 3, no. 4, pp. 3709–3716, Oct 2018.

[12] R. Hadidi, J. Cao, M. Ryoo, *et al.*, "Collaborative execution of deep neural networks on internet of things device," *arXiv preprint*, 2018.

[13] R. Hadidi, J. Cao, M. Woodward, *et al.*, "Real-time image recognition using collaborative iot devices," in *Proceedings of the 1st on Reproducible Quality-Efficient Systems Tournament on Co-designing Pareto-efficient Deep Learning*, ser. ReQuEST '18. New York, NY, USA: ACM, 2018.

[14] Y. Kang, J. Hauswald, C. Gao, *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *22nd ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2017, pp. 615–629.

[15] J. Hauswald, T. Manville, Q. Zheng, *et al.*, "A hybrid approach to offloading mobile image classification," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2014*. IEEE, 2014, pp. 8375–8379.

[16] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *37th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 328–339.

[17] Microsoft, "Embedded learning library (ell)," https://microsoft.github.io/ELL/, 2017, [Online; accessed 11/10/17].

[18] M. Rastegari, V. Ordonez, J. Redmon, *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV'16*. Springer, 2016, pp. 525–542.

[19] A. G. Howard, M. Zhu, B. Chen, *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[20] S. Han, H. Shen, M. Philipose, *et al.*, "Mcdnn: An execution framework for deep neural networks on resource-constrained devices," in *MobiSys'16*, 2016.

[21] Facebook, "Caffe2go: Delivering real-time ai in the palm of your hand," https://code.facebook.com/posts/196146247499076/delivering-real-time-ai-in-the-palm-of-your-hand/, 2017, [Online; accessed 11/10/17].

[22] Google, "Introduction to tensorflow lite," https://www.tensorflow.org/mobile/tflite/, 2017, [Online; accessed 11/10/17].

[23] F. N. Iandola, S. Han, M. W. Moskewicz, *et al.*, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[24] B. Asgari, R. Hadidi, H. Kim, *et al.*, "LODESTAR: Creating Locally-Dense CNNs for Efficient Inference on Systolic Arrays," *ACM/IEE Design Automation Conference (DAC) - Late Breaking Results, Las Vegas, NV*, 2019.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *26th Annual Conference on Neural Information Processing Systems (NIPS)*. ACM, 2012, pp. 1097–1105.

[26] J. Yu, A. Lukefahr, D. Palframan, *et al.*, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," in *44th International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 548–560.

[27] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations*. ACM, 2016.

[28] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplication," *arXiv preprint arXiv:1412.7024*, 2014.

[29] Y. Gong, L. Liu, M. Yang, *et al.*, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[30] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Proceeding Deep Learning and Unsupervised Feature Learning NIPS Workshop*, vol. 1. ACM, 2011, p. 4.

[31] iRobot Inc., "irobot create 2 open interface," www.cdn-shop.adafruit.com/datasheets create_2_Open_Interface_Spec.pdf, 2019, [Online; accessed 15/03/19].

[32] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[33] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[34] R. Hadidi, M. S. Cao, Ryoo, and H. Kim, "Robustly Executing DNNs in IoT Systems Using Coded Distributed Computing," *ACM/IEE Design Automation Conference (DAC) - Late Breaking Results, Las Vegas, NV*, 2019.