pCAMP: Performance Comparison of Machine Learning Packages on the Edges

Xingzhou Zhang, Yifan Wang
Wayne State University
Institute of Computing Technology, CAS
University of Chinese Academy of Sciences

Weisong Shi
Wayne State University

Abstract

Machine learning has changed the computing paradigm. Products today are built with machine intelligence as a central attribute, and consumers are beginning to expect near-human interaction with the appliances they use. However, much of the deep learning revolution has been limited to the cloud. Recently, several machine learning packages based on edge devices have been announced which aim to offload the computing to the edges. However, little research has been done to evaluate these packages on the edges, making it difficult for end users to select an appropriate pair of software and hardware. In this paper, we make a performance comparison of several state-of-the-art machine learning packages on the edges, including TensorFlow, Caffe2, MXNet, PyTorch, and TensorFlow Lite. We focus on evaluating the latency, memory footprint, and energy of these tools with two popular types of neural networks on different edge devices. This evaluation not only provides a reference to select appropriate combinations of hardware and software packages for end users but also points out possible future directions to optimize packages for developers.

1 Introduction

Recently, the burgeoning field of Machine Learning based applications has been enabled by the advances in three directions: ML models, processing power, and big data. In the cloud, developing new machine learning models needs not only numerous computational resources but also much time. In order to improve the efficiency of training models, many open-source machine learning packages have been recently developed, including TensorFlow [1] from Google, Caffe [2] from UC Berkeley, CNTK [3] from Microsoft, PyTorch [4] from Facebook, and MXNet [5] supported by AWS.

In the next five years, increasingly more data will be generated at the edge of the network collected by the Internet of Things, which in turn calls for edge computing [6]. Keeping data processing on the edges (such as edge servers, home server, vehicles, and mobile phones) is effective in offering real-time services [7, 8, 9, 10]. However, different from the data center in the cloud, edge nodes (also known as edge servers or fog nodes), usually have very limited computing power. Edge nodes are not a good fit for training ML-based models since they require a large footprint in both the storage (as big as 500MB for VGG-16 Model [11]) and computing power (as high as 15300 Million Multiply-Adds for executing deep learning models [12]).

To support executing large-scale models on the edges, several machine learning packages based on edge devices have been announced (Caffe2 for mobile [13], TensorFlow Lite [14]) which aim to dramatically reduce the barrier to entry for mobile machine learning. Unlike the training process on the cloud, machine learning models on the edges have been trained on the cloud, and the packages are designed to execute inference tasks on the edges.

However, few studies have evaluated these packages on edge devices. In this paper, we make a performance comparison of several state-of-the-art machine learning packages on the edges, including TensorFlow, Caffe2, MXNet, PyTorch, and TensorFlow Lite. We focus on their performance when running a trained model on the edges rather than the training process. We evaluate the latency, memory footprint, and energy of these packages with a large-scale model, AlexNet [15], and a small-scale model, SqueezeNet [16], on the edges. The devices are MacBook Pro (MacBook) [17], Intel's Fog Reference Design (FogNode) [18], NVIDIA Jetson TX2 (Jetson TX2) [19], Raspberry Pi 3 Model B+ (Raspberry Pi) [20], and Huawei Nexus 6P (Nexus 6P) [21].

Our major insights are summarized as follows: (1) There are no obvious single winners on every metric of all hardware platforms, but each has its merits. (2) The time required to load models is greater than it takes to run

Table 1: The overview of the combinations of hardware and software packages.

	MacBook Pro	Intel FogNode	NVIDIA Jetson TX2	Raspberry Pi	Nexus 6P
TensorFlow	√	√	√	√	×
Caffe2	√	√	√	×	×
MXNet	√	√	×	×	×
PyTorch	√	√	√	×	×
TensorFlow Lite	×	×	×	×	√

the model for most packages, which implies that there exist some opportunities to further optimize the performance. (3) Sacrificing space to improve execution efficiency can be a good method to optimize these packages.

The remainder of the paper is organized as follows. Section 2 presents the related work, and Section 3 introduces our experimental methods. Results and discussions are presented in Section 4. We make a conclusion and discuss our future work in Section 5.

2 Related work

In 2016, Shi [22] evaluated the performance of Caffe, CNTK, TensorFlow, and Torch and saw how they performed on cloud platforms, including two CPU platforms and three GPU platforms. This work focused on evaluating the training speed of these tools with fully connected neural networks (FCNs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), and benchmarking some distributed versions on multiple GPUs. Robert Bosch LLC presented a comparative study of five deep learning frameworks, namely Caffe, Neon, TensorFlow, Theano, and Torch, on three aspects: extensibility, hardware utilization, and speed [23]. In 2017, Louisiana State University analyzed the performance of three different frameworks, Caffe, TensorFlow, and Apache SINGA, over several HPC architectures in terms of speed and scaling [24]. It focused more on the performance characteristics of the state-of-the-art hardware technology for deep learning, including NVIDIA's NVLink and Intel's Knights Landing.

All of the research summarized above focused more on evaluating performance on the cloud. To our knowledge, our work is the first concerning the performance of machine learning packages on the edges.

3 Experimental Methods

3.1 Machine learning packages

Based on the popularity of these packages, this paper evaluates the performance of TensorFlow, Caffe2, MXNet, PyTorch, and TensorFlow Lite (as is shown in Table 1). Keras is also widely used; since it is built on top of TensorFlow, so we do not consider it.

TensorFlow [1] is developed by Google which has integrated most of the common units into the machine learning framework. It is an open source software library for numerical computation using data flow graphs. TensorFlow Lite is a lightweight implementation of TensorFlow for mobile and embedded devices. It enables on-device inference with low latency and a small binary size.

Caffe [2] is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors. Caffe2 is a deep learning framework that provides an easy and straightforward way to experiment with deep learning and leverage community contributions of new models and algorithms.

MXNet [5] is a flexible and efficient library for deep learning. It was initially developed by the University of Washington and Carnegie Mellon University, to support CNN and long short-term memory networks (LSTM).

PyTorch [4] is published by Facebook. It is a python package that provides two high-level features: tensor computation with strong GPU acceleration and deep Neural Networks built on a tape-based auto-grad system.

3.2 Machine learning models

Recently, convolutional neural networks (CNNs) have been deployed successfully in a variety of applications, including ImageNet classication [15], face recognition [25], and object detection [26]. Since the pre-trained AlexNet and SqueezeNet are widely available on many frameworks, we will use these two CNN-based models, AlexNet as the large-scale model and SqueezeNet as the small-scale model, to do evaluations.

AlexNet competed in the ImageNet Large Scale Visual Recognition Challenge in 2012 [27]. It has 61,100,840 parameters and 240MB size. Currently it is often used for evaluating the performance of hardware and software.

SqueezeNet [16] was proposed in 2016 as a small DNN architecture. It achieves AlexNet-level accuracy on ImageNet with 50x fewer parameters. It has 1,248,424 parameters and 4.8MB size.

Table 2: The experimental setup of hardware.

	CPU	Frequency	Cores	Memory	OS
MacBook Pro	Intel Core i5	2.7GHz	4	8GB	macOS 10.13.2
FogNode	Inter Xeon E3-1275 v5	3.6GHz	4	32GB	Linux 4.13.0-32-generic
Jetson TX2	ARMv8 + NVIDIA Pascal GPU	2GHz	6	8GB	Linux 4.4.38-tegra
Raspberry Pi	ARMv71	0.9GHz	4	1GB	Linux rasberrypi 4.4.34
Nexus 6P	Qualcomm Snapdragon 810	2GHz	8	2.6GB	Android 8.0 3.10.73

3.3 Edge devices

As is shown in Table 2, this paper adopts different kinds of hardware, including MacBook, FogNode, Jetson TX2, Raspberry Pi, and Nexus 6P. Their CPUs, memory, and operating systems are also different. Note that Jetson TX2 is an embedded AI computing device, it contains an ARM CPU and a NVIDIA Pascal GPU.

Table 1 lists the combinations of hardware and software packages that we were able to install successfully and get the experimental results in this paper. It is a big challenge to install these packages on different types of hardware. In order to ensure the fairness of the evaluation, the trained AlexNet and SqueezeNet models that we used were downloaded from the official websites (such as Caffe2 Model Zoo [28]). For TensorFlow, we only adopted AlexNet to do the experiments because the official website has not yet published a trained SqueezeNet model. For TensorFlow Lite, neither SqueezeNet nor AlexNet were officially available, so for TensorFlow Lite on Nexus 6p, we ran a pre-trained MobileNet [12] model.

3.4 Metrics

This paper evaluates the performance of packages when executing the inference tasks on the resource-constrained edges. Therefore, it leverages latency, memory footprint, and energy as the metrics.

This paper measures latency by using the "time" function in Python source scripts. Latency is divided into two metrics: inference time and total time. When packages are running the trained models, they should import the package first, and then load the model and image. Then it calls the function provided by the package to do the inference. To avoid errors, we called the inference function 100 times in each experiment. The inference time is the average time of the 100 inferences take. The time that the whole process takes is the total time.

For Nexux 6P, we monitor the memory footprint by leveraging the "Trepn" application developed by Qualcomm [29]. For Linux-based systems, which include FogNode, Jetson TX2, and Raspberry Pi, we get the memory footprint from the "ps" instruction. For MacBook, the "Activity Monitor" application provides each process's memory footprint.

For Nexus 6p, the energy can be obtained by leveraging the "Trepn" application. For the Linux-based operating system, we get the energy from the RAPL[30] tools on Linux. The power usage of each process cost can be monitored by RAPL. For MacBook, we use the "Intel Power Gadget" application to get the energy [31]. The app estimates the power/energy usage of all processes. To get the energy that just the package costs, we get a baseline first. We stop almost all service and wait until the power usage of the CPU is stable. Then we collect the power usage for 400 seconds. As is shown in Figure 1, the purple line represents the power usage when the CPU is idle, which is the baseline. Then, we start a process (such as run a pre-trained model) and collect the power usage also for 400 seconds. The green line shown in Figure 1 represents the power consumed by the process. Then we can estimate the energy that the process costs by calculating the area crossed by the two lines, as shown by the shaded part in Figure 1.

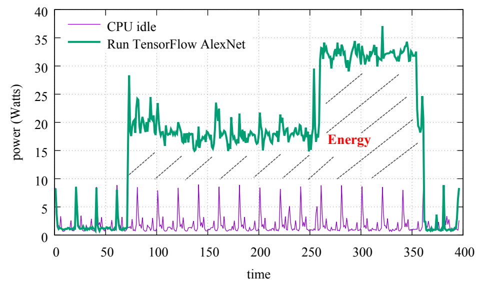
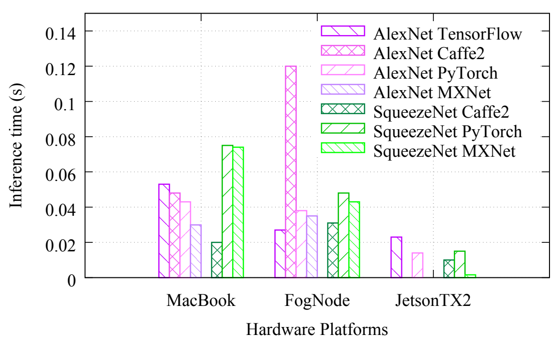


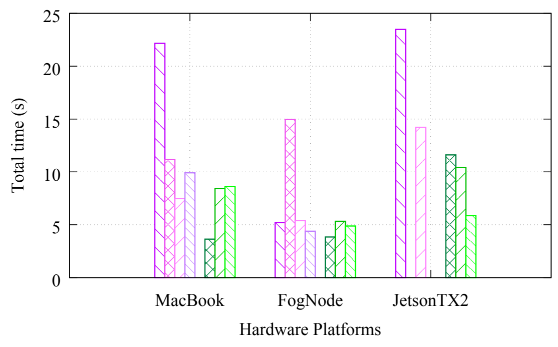
Figure 1: The method to obtain energy.

4 Results and Discussions

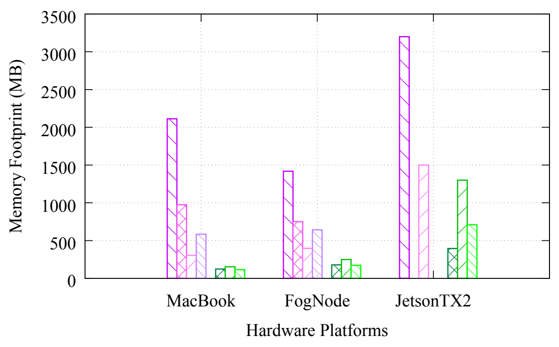
Figure 2 shows the results of MacBook, FogNode, and Jetson TX2. The results on Raspberry Pi and Nexus 6P will be presented directly in the text. The four figures have the same legends. Because of space limitations, we just plot the legend in Figure 2(a) and omit others. As is explained in Section 3, since the official website of TensorFlow does not provide trained SqueezeNet, we do not benchmark TensorFlow on SqueezeNet.



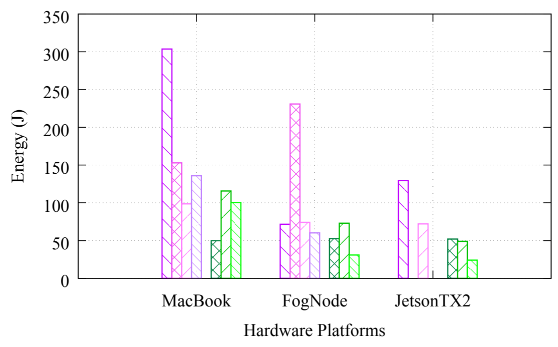
(a) Inference time



(b) Total time



(c) Memory footprint



(d) Energy

Figure 2: Performance comparison of Machine Learning Packages on the Edges

4.1 Latency

On AlexNet, as is shown in Figure 2(a) and Figure 2(b), MXNet has in general the best performance on MacBook. It only takes 0.029s to execute an inference task, which is 1/2 of TensorFlow's 0.053s. On FogNode, both PyTorch (0.038s) and MXNet (0.035s) perform slightly worse than TensorFlow (0.027s). Caffe2's performance on FogNode and Jetson TX2 are not good. The inference time of Caffe2 is around 4x longer than TensorFlow on FogNode. Due to install limitation, we only compare the performance of TensorFlow and PyTorch on Jetson TX2 when running AlexNet, PyTorch has a shorter inference time and total time than TensorFlow. On SqueezeNet, Caffe2 results in the best performance whose inference time is only about 1/4 of PyTorch and MXNet on MacBook. MXNet works slightly better than PyTorch. On FogNode, they have the similar latency. Each package takes less time on Jetson TX2 than MacBook and FogNode, especially for MXNet. The packages use optimized GPU code and so run faster on the Jetson TX2, the only system with a supported GPU.

Due to the resource constraints on Raspberry Pi, the inference time when running SqueezeNet on Caffe2 is 2.08s, and the total time is 218.67s. The inference time of Nexus 6P when running MobileNet based on TensorFlow Lite is 0.26s. Although it is much longer than running on MacBook, FogNode, and Jetson TX2, it is really great progress for running a model on small edge devices.

Table 3: Time profile of PyTorch on Jetson TX2

	AlexNet (s)	SqueezeNet (s)
import package	0.77	0.74
load model	**9.50**	**6.20**
first load image	0.12	0.15
first inference	**2.34**	**1.76**
second load image	0.075	0.07
second inference	0.006	0.019
inference 100 times	**1.35**	**1.45**
total time	14.17	10.41

To profile where the time is spent, we divide the whole process of running the trained model into seven parts and collect the time that each part takes. Table 3 represents the time of every part takes when it leverages PyTorch to execute the inference task on Jetson TX2. Doing the inference 100 times only needs 1.35s while loading model needs 9.50s when running AlexNet. It has similar performance when running SqueezeNet. In addition, we also find that doing the inference for the first time takes longer than the other 100 times for both AlexNet and SqueezeNet.

Insight 1: The time taken to load models is greater than what it takes to run the model for some packages, which implies that there exist some opportunities to further optimize the performance on the edges by decreasing the loading latency and frequency.

4.2 Memory footprint

On AlexNet, as is shown in Figure 2(c), the memory footprints on MacBook and FogNode are similar. TensorFlow occupies more than 2000MB memory, which is the largest. PyTorch performs the best as it has the least memory utilization, less than 500MB. On JetsonTX2, the memory footprint of TensorFlow is up to 3000MB while PyTorch utilizes only 1/2 of TensorFlow.

On SqueezeNet, the three packages have the similar memory use on CPU-base hardware, MacBook and FogNode. On Jetson TX2, PyTorch used 1301MB memory, which is 2x of MXNet (709MB) and 4x of Caffe2 (375MB). The memory of Caffe2 is 132MB when running SqueezeNet on Raspberry Pi. TensorFlow Lite occupies about 84M memory when running MobileNet on Nexus 6P.

Analyzing Figure 2, we find that these packages favor tradeoff between memory and latency. On AlexNet, the inference time of TensorFlow is much shorter than Caffe2 on FogNode, while it utilized more memory. On SqueezeNet, the total time of MXNet is 2x than Caffe2 while MXNet occupies 1/2 memory of Caffe2.

Insight 2: Based on the observation, we believe that MXNet may sacrifice space to improve efficiency, which can be a useful optimized method for executing real-time tasks on the edges.

4.3 Energy

On MacBook, Caffe2 could be regarded as the most energy-efficient package as it consumes the minimal energy. On FogNode, Caffe2 consumes about 2x energy of other packages when running AlexNet. PyTorch performs better than TensorFlow on JetsonTX2. MXNet has the best performance on FogNode and JetsonTX2 when running SqueezeNet. We take running AlexNet on FogNode as an example to profile the energy. Figure 3 shows the energy consumption of these packages. Based on the energy consumption, each process can be roughly divided into two periods: load models and inference. The power of the loading model is smaller than the inference, but the time that the loading model takes is greater than the inference for TensorFlow and MXNet.

Insight 3: Comparing Figure 2(c) and Figure 2(b), we can draw a conclusion that there is a positive correlation between energy and latency. Longer latency leads to a greater energy consumption.

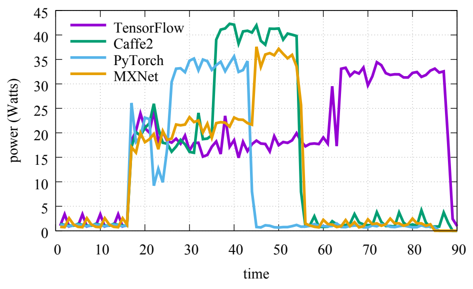


Figure 3: Energy profiling of AlexNet on FogNode

4.4 Summary

Figure 2 shows that there are no obvious single winners on every metric among the hardware platforms; each has its merits. We summarize the merits of every package and provide a reference to select appropriate packages for end users:

1) TensorFlow is faster when running large-scale model on the CPU-based platform FogNode, while Caffe2 is faster when running small-scale model. MXNet is the fastest package when running on Jetson TX2.

2) PyTorch is more memory efficient than other packages for MacBook and FogNode.

3) MXNet is the most energy efficient package on FogNode and Jetson TX2 while Caffe2 performs better on MacBook.

5 Conclusion

This paper aims to evaluate the performance of a set of machine learning packages when running trained models on different edge hardware platforms. By comparing the latency, memory footprint, and energy of these packages on two neural network models, AlexNet and SqueezeNet, this paper finds that there are no single packages that can consistently outperform others in both time and space dimensions. Also, the process of loading model takes more time for some packages than running the model, which implies that there exist some opportunities to further optimize the performance on the edges.

In the future, we plan to benchmark some optimized inference-only frameworks (such as Intel Computer Vision SDK [32]) and more hardware platforms (such as iPhone and Movidius Myriad VPU chip [33]). Also, we will try to propose a unified metric that can combine different metrics to evaluate these packages on the edges.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[3] Frank Seide and Amit Agarwal. Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2135–2135. ACM, 2016.

[4] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS workshop*, number EPFL-CONF-192376, 2011.

[5] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016.

[7] Mahadev Satyanarayanan, Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, Wenlu Hu, and Brandon Amos. Edge analytics in the internet of things. *IEEE Pervasive Computing*, 14(2):24–31, 2015.

[8] Qingyang Zhang, Yifan Wang, Xingzhou Zhang, Liangkai Liu, Xiaopei Wu, Weisong Shi, and Hong Zhong. Openvdap: An open vehicular data analytics platform for cavs. In *Distributed Computing Systems (ICDCS), 2017 IEEE 38th International Conference on*. IEEE, 2018.

[9] Kyungmin Lee, Jason Flinn, and Brian D Noble. Gremlin: scheduling interactions in vehicular computing. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 4. ACM, 2017.

[10] Bozhao Qi, Lei Kang, and Suman Banerjee. A vehicle-based edge computing platform for transit and human mobility analytics. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 1. ACM, 2017.

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

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

[13] Caffe2: A new lightweight, modular, and scalable deep learning framework, 2018.

[14] Introduction to tensorflow lite, 2018.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[17] Macbook pro. https://www.apple.com/macbook-pro/, 2018.

[18] Intel's fog reference design (intel fognode). https://www.intel.com/content/www/us/en/internet-of-things/fog-reference-design-overview.html, 2018.

[19] Nvidia jetson tx2 (jetson tx2). https://developer.nvidia.com/embedded/buy/jetson-tx2, 2018.

[20] Raspberry pi 3 model b+ (raspberry pi). https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/, 2018.

[21] Huawei nexus 6p (nexus 6p). https://consumer.huawei.com/en/phones/nexus-6p/, 2018.

[22] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking state-of-the-art deep learning software tools. In *Cloud Computing and Big Data (CCBD), 2016 7th International Conference on*, pages 99–104. IEEE, 2016.

[23] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. Comparative study of deep learning software frameworks. *arXiv preprint arXiv:1511.06435*, 2015.

[24] Shayan Shams, Richard Platania, Kisung Lee, and Seung-Jong Park. Evaluation of deep learning frameworks over different hpc architectures. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 1389–1396. IEEE, 2017.

[25] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[26] Jure Zbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, 17(1-32):2, 2016.

[27] Large scale visual recognition challenge 2012 (ilsvrc2012), 2018.

[28] Caffe2 model zoo, 2018.

[29] Trepn Profiler6.2: Qualcomm innovation center, inc., 2016.

[30] Spencer Desrochers, Chad Paradis, and Vincent M Weaver. A validation of dram rapl power measurements. In *Proceedings of the Second International Symposium on Memory Systems*, pages 455–470. ACM, 2016.

[31] Intel power gadget, 2018.

[32] Intel computer vision sdk - a brief overview. https://software.intel.com/en-us/blogs/2017/08/31/cv-sdk-a-brief-overview, 2018.

[33] Vision processing unita dedicated vision processing platform. https://www.movidius.com/solutions/vision-processing-unit, 2018.