Benchmarking of CNNs for Low-Cost, Low-Power Robotics Applications

Dexmont Pena, Andrew Forembski, Xiaofan Xu, David Moloney

***Abstract*—This article presents the first known benchmark of Convolutional Neural Networks (CNN) with a focus on inference time and power consumption. This benchmark is important for low-cost and low-power robots running on batteries where it is required to obtain good performance at the minimal power consumption. The CNN are benchmarked either on a Raspberry Pi, Intel Joule 570X or on a USB Neural Compute Stick (NCS) based on the Movidius MA2450 Vision Processing Unit (VPU). Inference performance is evaluated for TensorFlow and Caffe running either natively on Raspberry Pi or Intel Joule and compared against hardware acceleration obtained by using the NCS. The use of the NCS demonstrates up to 4x faster inference time for some of the networks while keeping a reduced power consumption. For robotics applications this translates into lower latency and smoother control of a robot with longer battery life and corresponding autonomy. As an application example, a low-cost robot is used to follow a target based on the inference obtained from a CNN.**

I. Introduction

Recent advances in machine learning have been shown to increase the ability of machines to classify and locate objects using Convolutional Neural Networks (CNN). For robotics applications this translates into increased reliability for tasks like tracking and following a target. However these networks typically have a large number of parameters resulting in a large number of operations.

In order to perform the large number of operations required by CNNs in real time, current systems typically use setups including expensive GPUs or distributed systems. This limits the application of CNNs on low-cost robots typically used by students and hobbyists.

Due to the popularity of CNNs in recent years a variety of different frameworks have been proposed, these include CuDNN (limited to Nvidia GPUs) [13], Caffe [14], MXNet [12] and Tiny-DNN [6] and more recently Googles TensorFlow [7] which is currently the most popular deep learning framework. Some of these frameworks have been shown to work on embedded platforms like Raspberry Pi [3] and the Intel Joule [2] but due to hardware limitations processing can be slow, taking a few seconds per frame even for custom designed networks [25].

One alternative to circumvent the hardware limitations of low-cost robots is to use a dedicated hardware accelerator like the Neural Compute Stick (NCS) [1]. This USB device offloads compute-intensive CNNs while still being low-cost and low-power. In this paper we benchmark different Convolutional Neural Networks (CNN) for image classification focusing on power consumption and inference times on embedded platforms. Then we show an application of a low-cost follower robot.

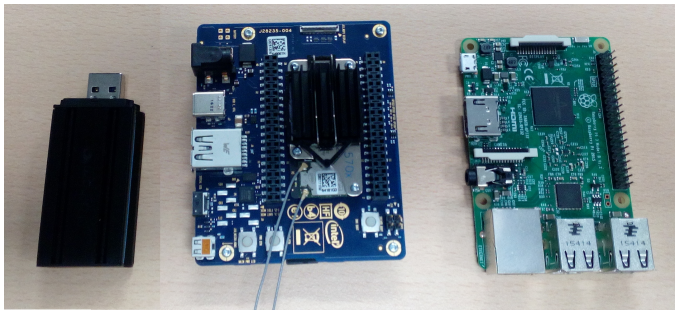

Fig. 1: Embedded environments used to evaluate the inference time and power consumption of the CNNs. From left to right: NCS, Intel Joule 570X, Raspberry Pi 3 Model B.

II. Related Work

Current benchmarks for Deep Learning Software tools limit the scope of their usage to the use of high power CPUs and GPUs and focus on the training time and accuracy [22]. No benchmarks have been found taking into account power consumption and running time on embedded systems. The only available benchmark focused on embedded systems evaluates only one network with a few layers[20] and provides no power consumption data.

Due to the processing power limitations of embedded platforms only a few alternatives exist that are able to perform inference on these low-power platforms. TensorFlow [7], Caffe [14] and MXNet [12] have been shown to compile on the Raspberry Pi [5] [20]. Tiny-DNN [6] was developed for low-resources in mind and is a header only library. Common libraries for image processing like DLib [15] and OpenCV [10] have been extended with modules to support DNNs however the number of networks they support is limited.

Only a few frameworks have been optimized for their use in low-cost low-power systems. Peter Warden optimized the DeepBelief SDK for the Raspberry Pi [25] however the current version requires three seconds to process a single frame using AlexNet [8]. Other libraries like OpenBLAS [24] focus on the Linear Algebra routines and can be used by some of the DNN frameworks to speed-up the processing. Caffe, Torch and MXNet include support for OpenBLAS to accelerate processing.

One of the main challenges for using the existing frameworks is the lack of standardization for storing the network

TABLE I: Software versions used for the evaluation of the DNN. TensorFlow is compiled with the flags "-mfpu=neon-vfpv4 -funsafe-math-optimizations -ftree-vectorize".

Software	GitHub Commit ID / Version
OpenBlas	92058a7
Caffe	f731bc4
TensorFlow	1.0.1
NCS	v1.02

TABLE II: Benchmarked networks. The models were downloaded from the Caffe and TensorFlow repositories.

Network	Layers	Operations
GoogLeNet [23]	27	~5 million
AlexNet [8]	11	~60 million
Network in Network [17]	16	~2 million
VGG_CNN_F [11]	13	~500 thousand
CIFAR10 [16]	9	~45 thousand

weights. Although some of frameworks offer parsers for Caffe [6], [10], [6], [12] the support for importing different network architectures is limited. The SDK for the NCS provides parsers for TensorFlow and Caffe, although with some limitations on the network architectures. Other alternatives for embedded Deep-Learning include FPGAs [19] but they can have a power consumption of up to 25W which limits their application for battery powered robots.

III. Experimental setup

Three platforms are used for evaluating the performance of the CNNs; the Raspberry Pi 3 Model B [3]; the Intel Joule 570X [2] and the Neural Compute Stick [1]. The Neural Compute Stick (NCS) is a low-cost and low-power USB device based on Myriad 2 MA2450 Vision Processing Unit (VPU) [9]. It supports loading networks designed and trained on common deep-learning frameworks like TensorFlow[7] and Caffe [14]. Fig. 1 shows the hardware used for running the CNNs.

TensorFlow and Caffe are used for performing the inference on the Raspberry Pi and the Joule, alternatively, the NCS is attached to each of the platforms over USB for offloading the inference. The power consumption is measured by using the INA219 power monitor IC from Texas Instruments. For the Raspberry Pi and the Intel Joule the INA219 is attached to the power line and then the power is measured before and while performing the inference and the difference is taken as the power required for performing the inferences. For the NCS, the INA219 is attached to the USB connection directly. The inference time is measured by using the system clock.

The version of the used frameworks is shown in Table I. Caffe is compiled with OpenBlas using the branch optimized_for_deeplearning. Tensor flow is installed by using the pip package manager. The Raspberry Pi is setup with Raspbian Jessie and the Intel Joule is setup with Ubuntu 16.04. Other frameworks compiled were MXNet, Tiny-dnn, OpenCV and DLib but during the testing they showed unstable behaviour or lack of support for the tested networks so they are not included in the evaluation. The networks used for the NCS were created using the Caffe models as input and setup to use 12 processing cores on Myriad 2 for the inference.

IV. Benchmarking

The models and weights for the evaluated networks were downloaded from the TensorFlow and Caffe repositories. The tested architectures are: GoogLeNet [23], AlexNet [8], VGG_CNN_F [11], CIFAR10 [16] and Network In Network (NiN) [17]. Table II show details of the tested DNN.

Figure 2 and 3 show the average time and power consumption for the classification on each of the networks for the Raspberry Pi 3, the Intel Joule 570X and the NCS. A batch size of 1 was used as it would be used in a real world application. The figures show that the fastest inference can be obtained on the Intel Joule at the cost of higher power consumption. The NCS provides a middle point in a trade-off between inference time and power consumption. The setup of the Raspberri Pi and Caffe provides the least power consumption at a slower inference time.

Fig. 2 shows that Caffe on the Raspberry Pi is the slowest performing combination while it is the one with the least power consumption. The inference on Joule with TensorFlow is the fastest at the cost of being the setup that requires the highest amount of power. Using the NCS provides a middle point in a trade-off between inference time and power consumption. As no tuning of the NCS is performed, it is expected that an optimized set of parameters results in a faster inference while maintaining the low power consumption.

V. Object Following

As an application of the CNN inference on low-cost robotics we setup a robot to follow a target autonomously. We use the Raspberry Pi to capture images from the PiCam and then the captured images are sent to the NCS to perform inference. We used the Sinoning Steering robot as it can be purchased on-line for less than $30USD [4]. Figure 4 shows our setup.

GoogLeNet is used to classify the contents of at template and decide if the shown image corresponds to the desired class as it showed a good accuracy and fast inference. The location of the target is determined by detecting a predefined template. The contents of the template are then send to GoogLeNet to perform inference and determine if the desired target is shown. The target class can be updated on the fly.

Once the desired target class has been recognized, its location and its size within the input image are used to compute the displacement of the robot. The robot is set to try maintain the located object at the centre of the input image and maintain it at a specific size, therefore it moves left/right based on the location of the object and forwards/backwards based on the area of the rectangle.

Table III shows the obtained runtime for each of the stages. The image preprocessing includes the capture of the frame, and image processing for determining the location of the target.

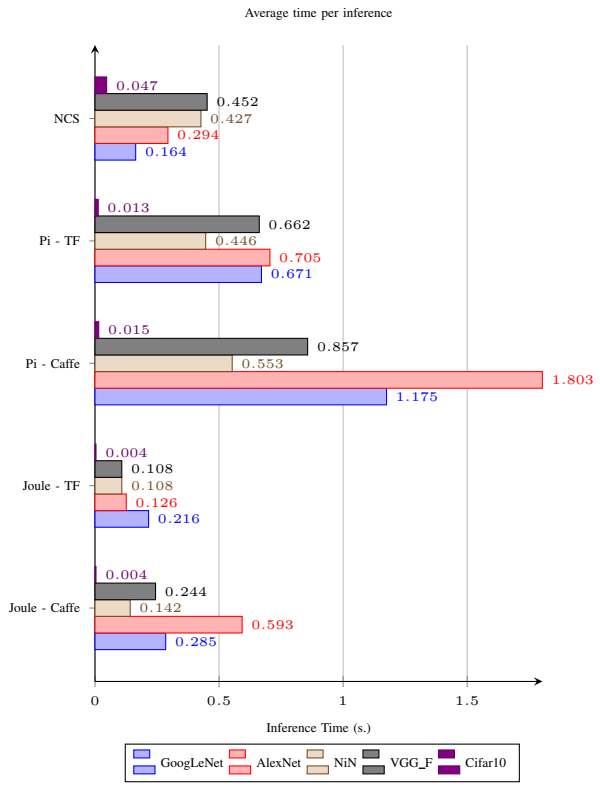

Fig. 2: Average time for a forward pass of the CNN using a batch size of 1. Pi - TF corresponds to a combination of the Raspberri Pi and TensorFlow. NCS corresponds to the time required for performing the inference and transferring the image over USB. Caffe was found to be highly sensitive to the number of parameters on the network while TensorFlow appears to be less sensitive.

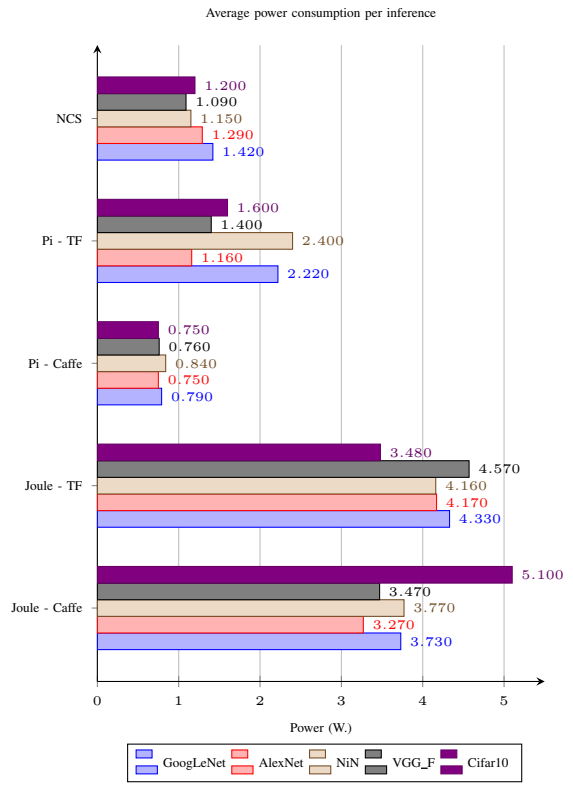

Fig. 3: Average power consumption during a forward pass of the CNN using a batch size of 1. The Intel Joule is shown to require the highest amount of power. The Raspberry Pi with Caffe requires the least amount of power.

TABLE III: Average times obtained for processing the images from the camera stream for TensorFlow and the NCS. The speed up obtained by using the NCS translates into processing frames at a rate of 4FPS which allows the quick control of the robot. This performance can not be achieved by using TensorFlow or Caffe on the Raspberry Pi.

Stage	Time (ms.)	
	NCS	TensorFlow
Image preprocessing	118	118
Classification	160	670
Total	278	788

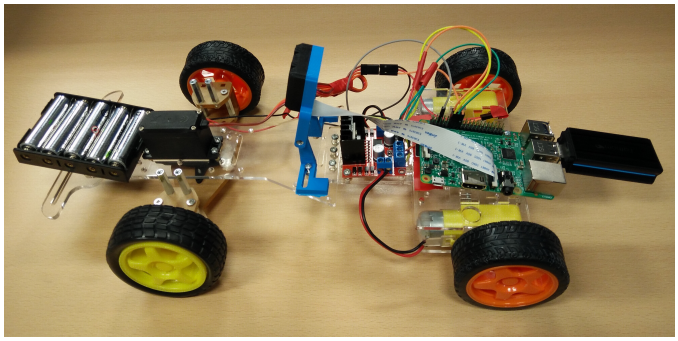

Fig. 4: Robot setup using the Sinoning steering robot, Raspberry Pi and NCS.

VI. Conclusions

This article presents the first benchmark of CNNs focused on inference time and power consumption for low-cost low-power robotics. We compared the performance of different CNNs using a variety frameworks and embedded platforms. The use of a dedicated hardware accelerator such as the NCS showed how to obtain an inference time up to 4x faster than using the embedded platforms alone while keeping a low-power consumption. As an application of this low-cost low-power environment we setup a robot to follow a target autonomously. By using the NCS we were able process a live stream at 4 FPS which was shown to be enough to allow smooth movement of the robot. Further research includes the evaluation of more CNNs like SSD [18] and YOLO [21] used for object detection. Also as the available frameworks provide support for more types of networks their performances in this kind of low-cost environment need to be tested.

References

[1] Movidius Announces Deep Learning Accelerator and Fathom Software Framework. URL https:

//www.movidius.com/news/movidius-announces-deep-learning-accelerator-and-fathom-software-framework.

[2] Intel Joule. URL https://software.intel.com/en-us/iot/hardware/joule/dev-kit.

[3] Raspberry Pi. URL https://www.raspberrypi.org/.

[4] Sinoning Steering. URL https://www.sinoning.com/collections/robot-car-chassis/products/steering-engine-4-wheel-2-motor-smart-robot-car-chassis-kits-diy-for-arduino.

[5] TensorFlow on the Raspberry Pi. URL https://github.com/samjabrahams/tensorflow-on-raspberry-pi.

[6] tiny-dnn. URL https://github.com/tiny-dnn/tiny-dnn.

[7] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. mar 2016. URL http://download.tensorflow.org/paper/whitepaper2015.pdfhttp://arxiv.org/abs/1603.04467.

[8] Krizhevsky Alex, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[9] Brendan Barry, Cormac Brick, Fergal Connor, David Donohoe, David Moloney, Richard Richmond, Martin O'Riordan, and Vasile Toma. Always-on Vision Processing Unit for Mobile Applications. *IEEE Micro*, 35(2):56–66, mar 2015. ISSN 0272-1732. doi: 10.1109/MM.2015.10. URL http://ieeexplore.ieee.org/document/7024073/.

[10] Gary Bradski. The opencv library. *Doctor Dobbs Journal of Software Tools*, 25(11):120–126, 2000.

[11] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In *Proceedings of the British Machine Vision Conference 2014*, pages 6.1–6.12. British Machine Vision Association, 2014. ISBN 1-901725-52-9. doi: 10.5244/C.28.6. URL http://arxiv.org/abs/1405.3531http://www.bmva.org/bmvc/2014/papers/paper054/index.html.

[12] 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. *eprint arXiv:1512.01274*, pages 1–6, 2015. URL http://arxiv.org/abs/1512.01274.

[13] Sharan Chetlur and Cliff Woolley. cuDNN: Efficient Primitives for Deep Learning. *arXiv preprint arXiv: ...*, pages 1–9, 2014. URL http://arxiv.org/abs/1410.0759.

[14] 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 ACM International Conference on Multimedia - MM '14*, pages 675–678, New York, New York, USA, 2014. ACM Press. ISBN 9781450330633. doi: 10.1145/2647868.2654889. URL http://dl.acm.org/citation.cfm?doid=2647868.2654889.

[15] Davis. E. King. Dlib-ml: A Machine Learning Toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009. ISSN 15324435. doi: 10.1145/1577069.1755843. URL http://jmlr.csail.mit.edu/papers/v10/king09a.html.

[16] Alex Krizhevsky and G Hinton. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, pages 1–9, 2010. URL http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:Convolutional+Deep+Belief+Networks+on+CIFAR-10{#}0.

[17] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. In *International Conference on Learning Representations (ICLR) 2014*, page 10. IEEE, dec 2014. ISBN 9781479972913. doi: 10.1109/ASRU.2015.7404828. URL http://arxiv.org/abs/1312.4400http://ieeexplore.ieee.org/document/7404828/.

[18] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9905 LNCS, pages 21–37. 2016. ISBN 9783319464473. doi: 10.1007/978-3-319-46448-0_2. URL http://link.springer.com/10.1007/978-3-319-46448-0{_}2.

[19] Kalin Ovtcharov, Olatunji Ruwase, Joo-young Kim, Jeremy Fowers, Karin Strauss, and Eric S Chung. Accelerating Deep Convolutional Neural Networks Using Specialized Hardware. *Microsoft Research Whitepaper*, pages 3–6, 2015. ISSN 1098-6596. doi: 10.1017/CBO9781107415324.004. URL http://research-srv.microsoft.com/pubs/240715/CNNWhitepaper.pdf.

[20] Dmytro Prylipko. How to run deep neural networks on weak hardware. URL https://www.linkedin.com/pulse/how-run-deep-neural-networks-weak-hardware-dmytro-prylipko.

[21] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 794:185–192, jun 2015. ISSN 01689002. URL http://arxiv.org/abs/1506.02640.

[22] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking State-of-the-Art Deep Learning

Software Tools. *arXiv:1608.07249 [cs]*, page 6, aug 2016. URL http://arxiv.org/abs/1608.07249.

[23] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9. IEEE, jun 2015. ISBN 978-1-4673-6964-0. doi: 10.1109/CVPR.2015.7298594. URL http://ieeexplore.ieee.org/document/7298594/.

[24] Qian Wang, Xianyi Zhang, Yunquan Zhang, and Qing Yi. AUGEM: Automatically Generate High Performance Dense Linear Algebra Kernels on x86 CPUs. *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 25:1—-25:12, 2013. ISSN 21674337. doi: 10.1145/2503210.2503219. URL http://doi.acm.org/10.1145/2503210.2503219.

[25] Pete Warden. Deep learning on the Raspberry Pi! URL https://petewarden.com/2014/06/09/deep-learning-on-the-raspberry-pi/https://github.com/jetpacapp/DeepBeliefSDK/.