An Evaluation of Energy-Efficient Devices for Real-Time Inference of Convolutional Neural Networks

Ajinkya Ghadge , Shashikant Ghangare and Nagaraj V. Dharwadkar[1]

***Abstract*— Significant advancements have been made in the field of image and video recognition using Convolutional Neural Networks. With their ability to generalize features and their remarkable accuracy, CNNs have performed very well with large image data-sets. Nonetheless, optimizing and deploying compute-intensive image recognition and detection CNNs to work in real-time on low-powered embedded devices is a significant challenge. We evaluate the performance of Raspberry Pi 3 Model B+, Intel Movidius Neural Compute Stick and Nvidia Jetson-TX2 board for energy-efficient and real-time inference performance on suitable CNN architectures. While the Jetson-TX2 board consumes 7.5 watts of power, the NCS has a small footprint of 1.2 watts. We also demonstrate the performance improvement in inference time and efficiency, when openvino and TensorRT optimizations are used for Intel NCS and Nvidia Jetson TX2 respectively. Based on our experiments, we present best suitable CNN architectures for real-time performance on the two embedded devices.**

I. INTRODUCTION

Recent developments in deep learning and convolutional neural networks, have significantly improved image classification and object detection accuracy of algorithms. Machine learning has found its way in a lot of application areas including Robotics and IoT. Despite an increase in the accuracy, these networks have a large number of parameters making the inference process compute-intensive, particularly for embedded devices.

Typically, to inference in real-time using these large number of parameters, expensive GPUs or distributed systems are used. IoT devices prefer to offload these compute-intensive tasks to a High-performance machine, often deployed on a cloud platform. This limits the application of CNNs when using with embedded devices, which use very little power and use limited bandwidth. Additionally these devices are constrained by the need to operate with very-low latency in a distributed environment.

Some of the software frameworks like Caffe and Tensorflow have been ported to work on embedded platforms like Raspberry Pi and Intel Joule, but processing can be slow due to limitations of hardware. To solve these challenges, there are energy-efficient inference platforms like Intel Movidius Neural Compute stick and Nvidia Jetson TX2, which have dedicated hardware built to suit the needs of deploying deep neural networks. The Jetson TX2 board consumes 7.5 watts of power and the NCS consumes 1.2 watts.

We present benchmarks for the two devices and optimal CNN architectures for making real-time predictions. We also measure the energy efficiency with respect to making real-time predictions on the two-devices to conclude the suitability of the devices for various embedded applications. Additionally, we demonstrate the use of TensorRT and OpenVino based software optimization to improve inference performance and energy efficiency on the two devices.

*This work was not supported by any organization

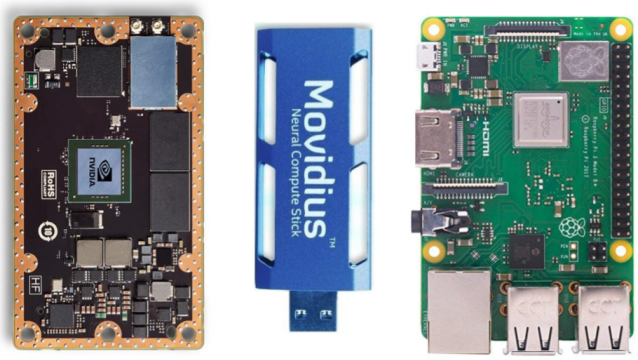

Fig. 1. Embedded devices used: Jetson TX2 Module, Intel Movidius NCS and Raspberry Pi 3 B+

II. RELATED WORK

A benchmark for power consumption on low-power embedded devices for Caffe and Tensorflow is given in [1]. Experimental setup to measure power consumption by the NCS, Raspberry Pi and Intel Joule board for a forward pass on AlexNet, GoogLeNet and VGG architectures for image classification is also explained. However, the study lacks measurements on optimized CNN implementations for real-time vision applications. Additionally, it doesn't include any major detection architectures.

Existing CNN inference benchmarks do not take into account the special case of real-time performance. Neither there is a direct comparison of Nvidia Tegra based Jetson TX2 and Intel Movidius VPU based NCS, nor has power consumption been tested in relation to frames per second achieved in the case of video. Additionally, there is no analysis of using software optimization for obtaining real-time performance.

[2]Another work, featuring first comprehensive technical overview of the VPU(Vision Processing Unit) in the context of Intel NCS. The paper explains the architecture and organization of the Intel NCS, the motivations behind creation of the NCS API and the principles used for off-loading of inference using a USB port to the NCS. There is also a comparison of inference on CPU, GPU and the VPU for accuracy. However it lacks comparison of other

contemporary inference platforms for power consumption and achievable frame rates for CNN architectures.

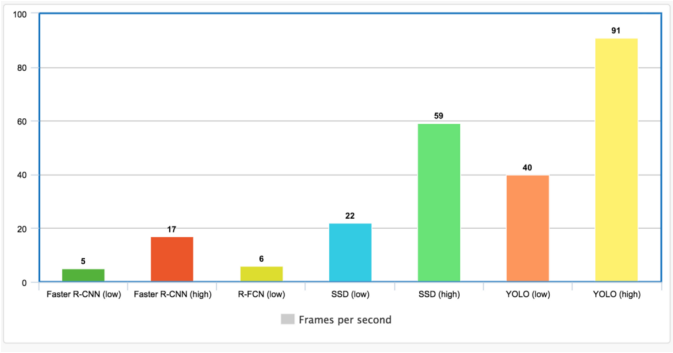

Fig. 2. Highest and Lowest FPS reported by Corresponding papers

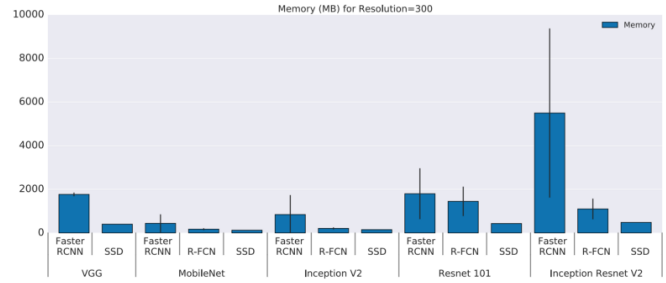

Fig. 3. Memory (Mb) usage for each model. Note that we measure total memory usage rather than peak memory usage. Moreover, we include all data points corresponding to the low-resolution models here. The error bars reflect variance in memory usage by using different numbers of proposals for the Faster R-CNN and R-FCN models (which leads to the seemingly considerable variance in the Faster-RCNN with Inception Resnet bar).

Comparison of Faster-RCNN, R-FCN and SSD object-detection architectures provides a guide for selecting a detection architecture that achieves speed/memory/accuracy balance for a given application. The work also investigates various ways to trade accuracy for speed and memory usage in modern convolutional object detection systems. The one of a kind study successfully compensates for different base feature extractors, different default image resolutions as well as different hardware and software platforms. However, it not only lacks a comparison and analysis with state of the art Yolo object-detection architecture but also does not benchmark on an embedded device.

III. EXPERIMENTAL SETUP

A. Hardware

With an embedded system-on-module (SoM) containing dual-core NVIDIA Denver2 + quad-core ARM Cortex-A57, 8GB 128-bit LPDDR4 and integrated 256-core Pascal GPU the NVIDIA Jetson TX2 is useful for deploying computer vision and deep learning tasks. The Jetson TX2 runs Linux and provides greater than 1TFLOPS of FP16 compute performance in less than 7.5 watts of power. It supports most major deep learning frameworks like Caffe, Tensorflow, MXNet and Torch.

The Intel NCS is a tiny, fan-less USB device that can be used with a host machine only to make prediction/inference. It is based on Myraid 2 MA2450, also referred to as VPU(Vision Processing Unit). Designed as a 28-nm co-processor, Myraid 2 VPU provides high-performance tensor acceleration. The chip dissipates less than 1W. The Myraid 2 VPU features 12 highly-parallelizable vector processors, name Streaming Hybrid Architecture Vector Engines(SHAVE). Each of these can be operated independently. Applications communicate with the VPU using the USB 3.0 Interface. The NCS SDK provides parsers only for Caffe and Tensorflow, with some limitations on the network architecture. Python and C++ based API is provided to perform these tasks.

B. Software

The Intel Movidius Neural Compute SDK (NCSDK) is used in two distinct modes to support development and deployment. The Full SDK mode installs both the toolkit and API framework on the host device. The installation process is slower but it provides the tool-chain to generate a graph file, that can be later used for deployment. Full SDK can also be used for profiling of the neural networks. The API-only mode provides an API framework in Python and C++, that can be used for loading the generated graph file, sending images to the device and making predictions on the sent images. Graph file, is a file containing weights optimized by using the NCSDK. Based on dependencies, API-only mode can be compiled on various embedded platforms. As a host for the NCS, we are using a laptop device with Intel i7-7700 running Ubuntu 16.04.4. Role of host machine is limited to sending data to the NCS, the forward pass is entirely computed on the NCS and similar performance can be expected when using any other device with NCS connected via a USB 3.0.

Openvino, extends computer vision (CV) workloads across Intel hardware, maximizing performance. It supports execution on Intel Movidius Neural Compute Stick and provides optimized calls for computer vision standards including OpenCV, OpenCL, and OpenVX.

The Jetson TX2 runs an optimized version of Ubuntu 16.04. There is no additional software that needs to be installed for inference on the Jetson TX2. Caffe is already compiled in the OS provided. Therefore, the weights file and the network model file can be used to make inference. Nvidia provides software optimization for inference in form of TensorRT which is also a part of the OS provided for the Jetson TX2. Add this here without using ”we” Each of these can be operated independently. Applications communicate with the VPU using the USB 3.0 Interface.

For development we are using Ubuntu 16.04 on a x86 architecture machine. We are able to compile the NCSDK in Full SDK mode.

Write this in tabular for as specific setup when working with NCS For working with the NCS, the batch size has to be set to 1, and the number of inupts to the network should also be 1. This is a constraint specified by the SDK. Also, there is no support for input parameters in the input layer. This

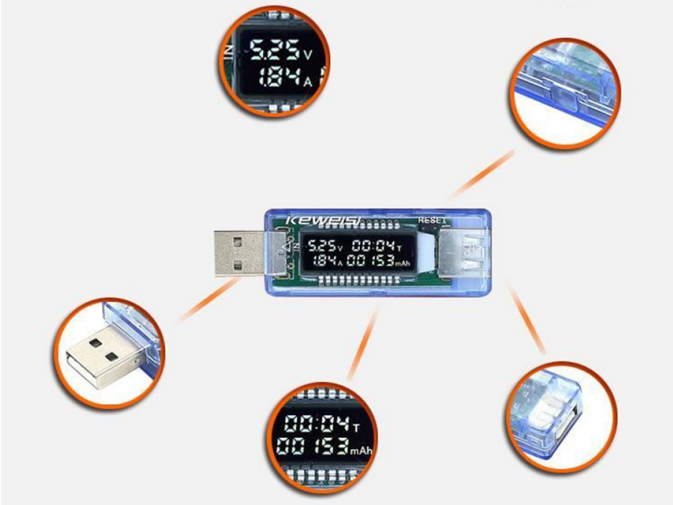

Fig. 4. Keweisi KWS-V20 USB Power Doctor and its components

should be changed to input shape. Input name in the input layer should always be "data". There is also a limitation on the size of graph file, which should be less than 320 MB. Additionally, the Intermediate layer buffer size should be less than 100 MB and scratch memory size should be less than 112 KB. Unlike the NCS, there are no changes that are needed to run the CNNs on the Nvidia Jetson TX2 platform.

For our benchmark we have chosen AlexNet and GoogLeNet as they are standard, well-documented architectures that have shown promising results on a wide range of data-sets. We have also included SqueezeNet, considering the memory limitations of the NCS. SqueezeNet generates 50 times lesser parameters as compared to AlexNet with same accuracy. We have trained these architectures on the ImageNet dataset. For object detection, we have chosen SSD on MobileNet considering the trade-of between speed, accuracy and memory needed. Fig 1 shows the highest and lowest FPS reported by the corresponding papers. Although not an apples to apples comparison due to different mAP, it gives a fair idea of achievable FPS, and suitable architectures for object detection. Fig 2 shows the average memory usage for each model. SSD on MobileNet has the lowest footprint.

C. Power Measurement

To account for power consumption, which is critical for embedded devices. While the Jetson TX2 has an inbuilt mechanism to comprehensively measure power consumption, we use an USB Charger Doctor to measure power consumption of the Intel Movidius NCS.

1) Power Measurement on Jetson TX2: The TX2 module has two INA3221 power monitors the carrier board has two more INA3221. The system on module includes three power rails VDD-IN,VDD-GPU and VDD-CPU. Since we only need to measure the power consumed by the system on module(SoM), we will only measure power on the power rail VDD-IN. Other rails that the VDD-IN powers are VDD-SYS-CPU, VDD-SYS-GPU, VDD-SYS-DDR, VDD-SYS-SOC and VDD-4V0-WIFI. Since we are turning the wifi module off, there is no power consumption by the VDD-4V0-WIFI. VDD-SYS-CPU and VDD-SYS-GPU are rails for CPU and GPU, while VDD-SYS-DDR and VDD-SYS-SOC are rails for ram and system on chip. Hence, measuring power consumed can be measured by using the rail VDD-IN. The shipped OS provides a python script named JetsonTX2power.py to measure voltage and current reading on different rails.

2) Power Measurement for Intel Movidius NCS: Previous attempts to measure power consumption on the NCS have used INA219 IC for power measurement. While the IC is suitable for measurement of power on a variety of embedded devices, we needed a USB interface to measure the power consumption on the NCS, which is not readily found for the IC. Alternatively, we have used the USB charger doctor, [3]Keweisi KWS-V20. The device comes with an OLED screen that displays the current and voltage values. The voltage range for the device is 4-20V and the current measurement range is 0-3A. Additionally, charge up to 99,999 can also be measured. The device contains a USB male connector that connects to the power source and a USB female connector that can be used to connect other devices, in our case the NCS. Since the NCS voltage and current value fall within the range of this device and its self consumption is meager 3.3 mA, it is a convenient choice to measure NCS power consumption.

IV. Benchmarking

Since, we are using standard CNN architectures and datasets, the accuracy for each of these is defined. There are no changes to the architecture, although optimization of the models is done using openvino and tensorrt only to improve inference time. Hence classification and detection accuracy of the chosen architectures does not change. There are three power operating modes on the Jetson TX2 board Max-Q, Max-P and Max-Clock. While the Dynamic Voltage and Frequency Scaling (DVFS) permits Jetson TX2s Tegra Parker SoC to adjust clock speeds at run time according to user load and power consumption, the Max-Q configuration sets a cap on the clocks to ensure that the application is operating in the most efficient range only. Max-P, the other preset platform configuration, enables maximum system performance in less than 15W. Max-Clock gives the best performance with highest power consumption.

The optimizations of TensorRT and Openvino are subject to the various layers supported by these frameworks. Currently few of the layers used in SSD and Yolov2 are not supported by TensorRT and Openvino. Some people have demonstrated possible work arounds by using custom layers, or replacing unsupported layers with alternative supported layers. These techniques are unsuitable for our benchmark process. This leaves us with no hardware specific optimization for Object detection architectures in this benchmark.

A. Measuring Efficiency

Frames/Second is often used as a measure of performance with respect to real-time scenarios in computer vision. Real-

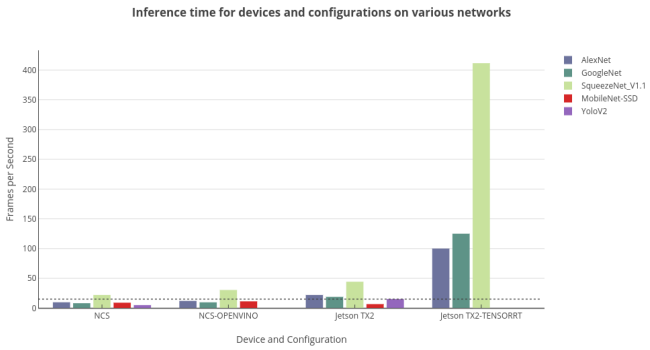

Fig. 5. Inference time for various devices and architectures

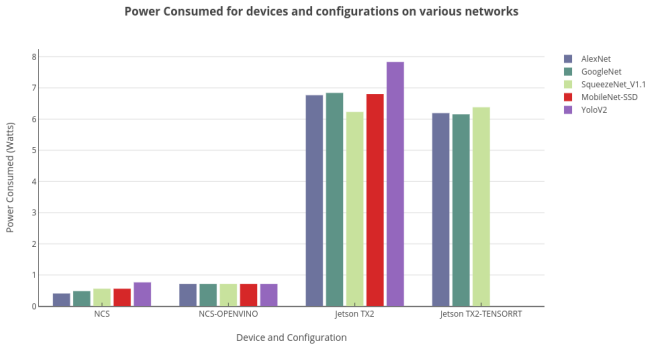

Fig. 6. Power consumption for various devices and architectures

time performance is said if a video stream can be processed between a range of 15-30 FPS or more. For measuring efficiency, we measure average power consumed during inference. The final measure of efficiency can be frame rate per unit power, which would be frames/second/watt.

B. Benchmark

The Jetson TX2 with tensorRT optimization performs the best for image classification tasks. Very high frame-rates are achieved for all the classification networks. There is a small difference in the power consumption, but overall the Jetson TX2 achieves significantly higher frame rate. The Intel NCS consumes significantly less power, but real-time performance is achieved only for SqueezeNet architecture. SSD-MobileNet comes close to real-time performance.

We also found in our observations that the Openvino based optimization are not as significant as the TensorRT.

V. CONCLUSION

Jetson TX2 with tensorRT is highly suitable for high-performance real-time tasks. It can be used for real-time inference of deeper CNN architectures like GoogLeNet. The Intel NCS shines when using SqueezeNetv1.1 for classification tasks and MobileNet-SSD for detection. For SqueezeNetv1.1, it achieves a balance of efficiency and practical usage with value of 42.57 images/sec/watt. For applications requiring 15-20 FPS classification rate and very less power consumption, the Intel NCS should be preferred. For tasks requiring higher accuracy, where larger CNNs are used, the Jetson TX2 can be used.

VI. PERFORMANCE IMPROVEMENT ON RASPBERRY PI

For comparison with previous benchmark of FPS achieved on Intel Movidius on Raspberry Pi, we look at the performance achieved for same neural architecture and the best FPS we achieved for same classification accuracy. Some important considerations here are that we have used Neural Compute Stick SDK-2 as compared to the SDK-1 used by the previous authors, there are no performance related changes documented.

Architecture	*GoogleNet (previous benchmark)*	*GoogleNet (our benchmark)*	*SqueezeNetv1.1 (best classification)*
FPS	6.25	7.75	19.12

VII. FUTURE WORK

With the pace of current research, new image classification and object detection architectures are coming up. Newer family of architectures like SquezeNext, MobileNetv2, YoloV3 have shown improvement in both accuracy and inference speed. Not only can the NCS be used in parallel with each other, NCS can also be used in tandem with the Jetson TX2 as it is possible to compile the SDK on the Jetson TX2. Frame-rates can also be improved by efficient memory management techniques and concurrency of CPU and GPU by using double-buffering. Finally, with custom-layer implementations, un-supported CNN layers in YoloV3 and MobileNet-SSD can be optimized using TensorRT.

REFERENCES

[1] D. Penna, A. Forembski, Xiaofan Xu, D. Moloney Benchmarking of CNNs for Low-Cost, Low-Power Robotics Applications, 2017, RSS 2017 Workshops

[2] Exploring the Vision Processing Unit as Co-Processor for Inference Sergio Rivas-Gomez, Antonio J. Pena, David Moloney, Erwin Laure, Stefano Markidis, 2018 IEEE International Parallel and Distributed Processing Symposium Workshops

[3] https://goughlui.com/2016/08/20/review-teardown-keweisi-kws-v20-usb-tester/