Developing IoT Devices Empowered by Artificial Intelligence: Experimental Study

Viktor Prutyanov, Nikita Melentev, Daniil Lopatkin, Alexander Menshchikov, Andrey Somov
Center for Computational and Data-Intensive Science and Engineering (CDISE)
Skolkovo Institute of Science and Technology
Moscow, Russia
viktor.prutyanov@skoltech.ru, nikita.melentev@skoltech.ru, daniil.lopatkin@skoltech.ru,
aleksandr.menshchikov@skolkovotech.ru, a.somov@skoltech.ru

***Abstract*—The number of real-world Internet of Things (IoT) deployments continuously and steadily increases while the capabilities of single IoT devices cannot yet be exploited for the purpose of Artificial Intelligence. Indeed, computation complexity and energy consumption are the constraining requirements for the development and implementation of truly intelligent IoT devices with AI. In this paper we present an experimental study where we perform the observation of time of execution, overheating, Central Processing Unit (CPU) load and power consumption using an embedded system by itself and embedded system with an external low-power Graphics Processing Unit (GPU) able to run the pre-trained neural networks for object detection. We report on the series of experiments conducted on the neural networks with different depth and input size. Experimental results demonstrate high potential of AI application on the IoT devices.**

***Index Terms*—Embedded systems, artificial intelligence, neural networks, experimental evaluation, Internet of Things**

I. Introduction

The Internet of Things (IoT) paradigm [1] allows for joining of myriads of devices, objects, and services while introducing intelligent capabilities in the swarm of these 'things' [2]. IoT has penetrated in a huge number of real-world applications within the last decade including smart-x applications (where x is a city, home, transportation), education, agriculture [3]. Indeed, lots of smart sensing solutions have become available on the market due to research solutions in wireless communication, sensing and actuation, low-power sensing and many others.

At the same time, the recent progress in Machine Learning (ML) clearly shows that there are still cases where the research community cannot rely on the powerful ML algorithms, e.g. based on neural networks, in the context of their application to the resource-constrained IoT devices which are supposed to be deployed everywhere and be available 24/7. The reason is the lack of energy stored in the battery-powered IoT devices and computation capability of 'things' which are expected to be deployed everywhere and be available any time. In fact, many research projects tackle the long-term operation problem of low-power sensing devices, e.g. sensor nodes, from a different perspective [4, 5, 6]. It is worth noting that most sensing devices are based on a low-power Micro Controller Unit (MCU) and fairly computational extensive algorithms could be run [7]. However, the research in adopting artificial intelligence for energy constrained devices, e.g. embedded systems and mobile phones, is still fragmented [8] with the limited number of experimental-based research projects [9].

A solution to address this problem could be to use an extra GPU for executing these algorithms. In our work, we use Intel Movidius Neural Compute Stick (NCS) with Raspberry Pi (RPi) embedded system.

For assessing the difference between RPi and NCS performance we use a set of different neural networks for object detection and classification. The following measurements fall in our experimental work: FPS (frames per second - number of processed images per time unit) that the system can achieve, CPU load, CPU temperature, RAM load, power consumption.

In this paper we assess the power consumption of RPi, which executes different neural network architectures, with the bundle of RPi and Movidius NCS. For one of the architectures, MobileNet v1, we also add a bundle of RPi and two NCSs to the comparison.

This paper is organized as follows: in Section II we introduce the reader to the experimental methodology and requirements, in Section III we provide and discuss our experimental results. Finally, we provide concluding remarks in Section IV.

II. Experimental Methodology

In this section, we present the experimental testbed and experimental methodology.

In all our tests we used the following software:

- Raspbian Stretch,
- Python 3.5,
- TensorFlow,
- Movidius Neural Compute SDK v2.

We use RPi 3 model B as a host and Intel Movidius NCS, connected to it via USB 2.0 interface. As the source of images for neural networks, the connected webcam is used. To measure current and therefore power consumption, we installed an ammeter in the power supply circuit of the controller or of the USB-hub. A 5-volt DC source is used to power the system.

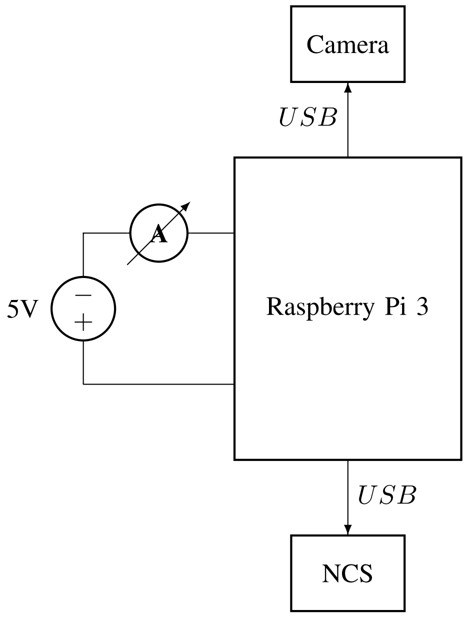

Fig. 1. Testbed scheme

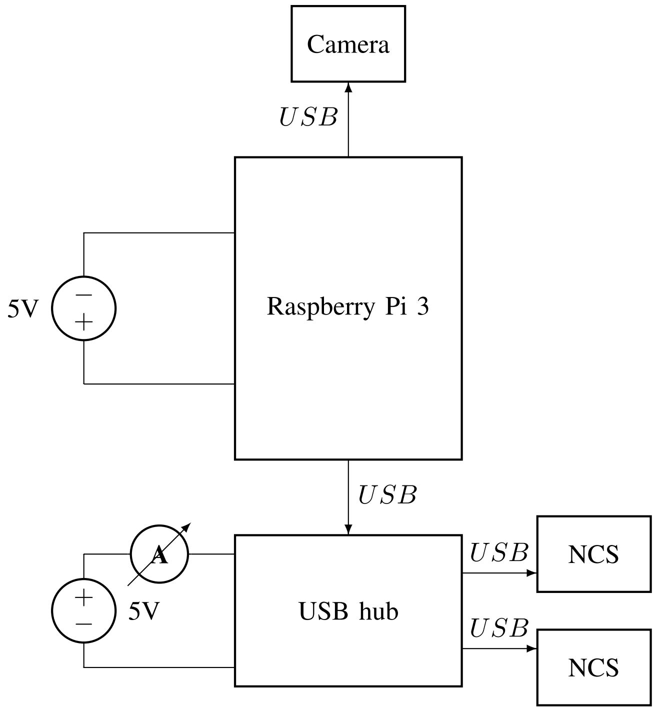

Fig. 2. Testbed scheme with USB hub

In our experiment, we run three neural network architectures: Inception v1[10], Tiny-YOLO v2[11] and MobileNet v1 [12]. Inception and MobileNet solve the classification problem with 1000 classes. Tiny-YOLO solves object detection problem with 20 classes. The environment was launched in two configurations. In the first configuration (RPi in the table) all computations are made on RPi. In second configuration (RPi + NCS in the table), Intel NCS takes the input image, executes the neural network and sends the output back to RPi. The corresponding scheme is shown in Fig.1.

MobileNet v1 architecture was also launched in the third configuration using two Intel NCSs, connected to RPi via the USB hub as it is shown in Fig.2. In this case, we have measured the current in the USB hub power circuit.

Movidius can only work with the graphs that can be converted from the TensoFlow or Caffee models. Pre-trained TensorFlow models were used. Due to the limited space we provide the detailed instructions on MobileNet in [13].

III. Experimental Results

In this section, we report on the experimental results and discuss them. In particular, we focus on the following metrics FPS, CPU and RAM load, CPU Temperature and discuss their dependence on the complexity of running neural networks.

In this case, we use MobileNet v1 set of neural networks which are different in terms of the depth and image input size. Depth is encoded in four classes: 0.25, 0.5, 0.75 and 1, each class corresponds to the accuracy which can be achieved and the number of iterations in this network. Figure 3 demonstrates that the greater depth parameter the better accuracy can be achieved. However, extra computations are required in this case.

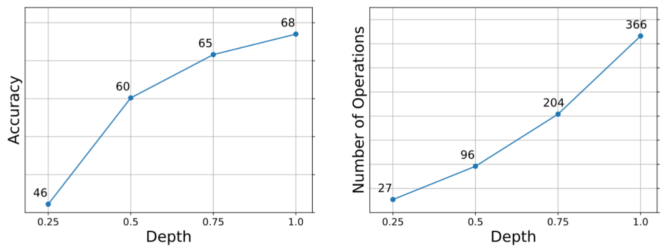

Fig. 3. MobileNet classification accuracy and computational complexity with different depths.

Similar situation can be observed with the input size. It is encoded in four classes: 128, 160, 192 and 224 pixels. All input sizes are square. With the increase in the input size, the accuracy increases with the number of iterations as it is shown in Figure 4.

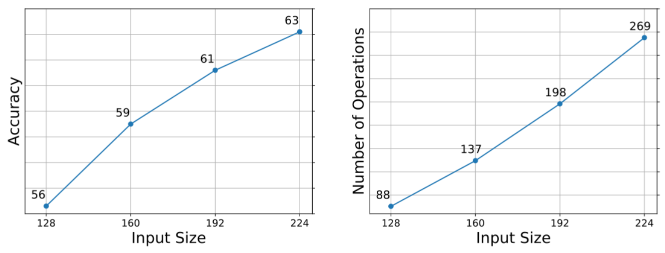

Fig. 4. MobileNet classification accuracy and computational complexity against different input sizes.

In these figures, we use the combined notation of the model depth and the input size.

A. FPS

Figure 5 shows the FPS value dependency on the parameters of executable neural networks. In this case, the pre-processed frames are used, so that we could exclude the frame processing time while computing the FPS values.

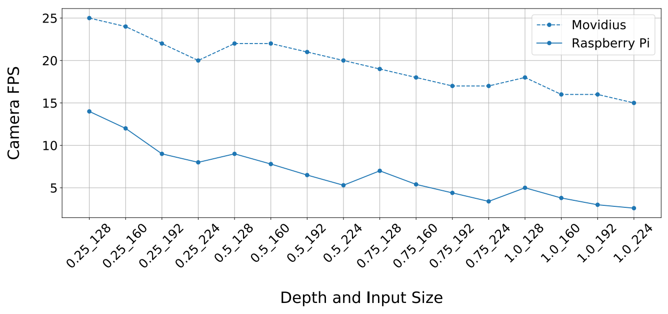

Fig. 5. Different MobileNet FPS rates with and without Movidius. Pre-processing time is excluded.

For the majority of experiments, the Movidius FPS is around 20, while the mean value of RPi FPS is 7. In general, for an object detection related applications 1 FPS is a reasonable result. That is why RPi can be used even for stronger MobileNet neural networks in terms of a frames-per-second metric.

B. CPU and RAM Load

Next, we consider CPU and RAM load are shown in Figure 6 and Figure 7, respectively. In real applications, in addition to running camera with the neural networks predicting labels, the user will most likely run other processes, e.g. storing the data to a database or run other system scripts. It is highly important taking this opportunity into account: only running a heavy neural network on RPi can take up almost all of the CPU power and a considerable portion of RAM.

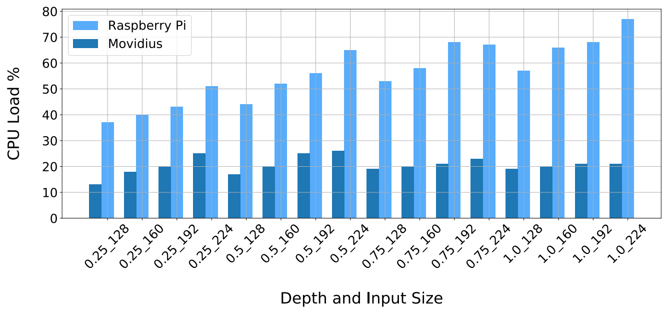

Fig. 6. Raspberry Pi CPU load during MobileNet execution on Raspberry Pi and Movidius NCS.

Usage of Movidius does not affect CPU functioning and only 20% is used for the image pre-processing task. On the contrary, using only RPi takes an additional 20% to 60% of CPU usage depending on the neural network size.

Figure 7 shows a comparative study on RAM Load while using the lightest and the heaviest models. Movidius requires the same amount of memory for both cases, which is around

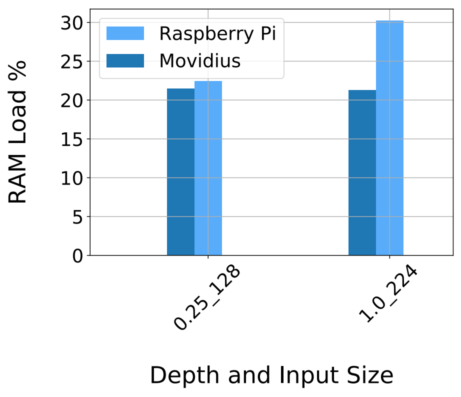

Fig. 7. Raspberry Pi RAM consumption during the MobileNet execution on Raspberry Pi and Movidius NCS.

22%. If using only RPi, we will need the extra 10% when launching a heavier model.

C. CPU Temperature

The most critical part of our experiments is to figure out whether we can safely execute the neural networks for an extended period of time. If the IoT device has sufficient power supply, the only thing that we have to avoid is overheating. CPU Temperature analysis while using only RPi and Movidius is depicted in Figure 8.

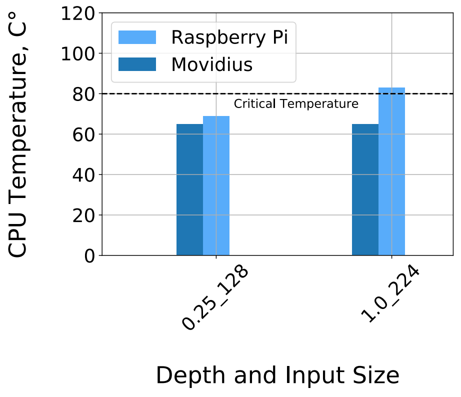

Fig. 8. Raspberry Pi CPU package temperature during the MobileNet execution.

Once again we compare the CPU Temperature with the lightest and heaviest models on board. Running Movidius heats up the CPU up to 64 in both experiments. However, executing the same neural networks only on RPi results in overheating in the second experiment with the largest model. It means that if we continue this example over the extended period of time, we will face the problem of performance reduction, increased power consumption and in far prospective the CPU quality reduction.

In the case with MobileNet v1 set of neural networks, we can execute up to the depth equal to 1 and input size equal to 160 models on RPi without Movidius. Heavier models (with the depth 1 and input sizes 192 and 224 pixels) result in overheating and we have to use Movidius.

D. *FPS and Power Consumption*

In all cases, a neural network takes 300 frames from the camera in real time. For Inception v1 and MobileNet v1 frames are being resized to $224 \times 224 \times 3$. For Tiny YOLO v2 frames are being resized to $416 \times 416 \times 3$. In this case, we include the frame processing time and afterward we compute the FPS value.

We have recorded the time steps which determine the moment of end of initialization and start of neural network. Initialization phase proceeds on RPi and includes loading computational graph in RAM, the NCS initialization and loading graph to the NCS. Processing phase includes sending of the resized frames to NCS and receiving the corresponding class number or class number and bounding box coordinate respectively.

We have measured the current flowing through RPi for both experimental environment power circuit and provide several characteristics.

- FPS denotes the number of frames processed per time unit during the Processing Phase.
- Initialization time denotes the Initialization Phase duration from application start to the Processing Phase start.
- Average Current denotes the average current during the Processing Phase.
- Power Consumption denotes average power consumption during Processing Phase.
- Power Efficiency denotes the FPS processed per watt.

MO label, in case of current, power consumption and efficiency, denotes that the value was measured only for NCS.

First of all, we run Inception v1 on Raspberry PI with NCS (see 9).

We were unable to run Inception v1 on single RPi because 1Gb of RAM is not enough that is why there are no measurements for RPi in Table I.

TABLE I
INCEPTION V1

	RPi	**RPi + NCS**
FPS	N/A	5.82
Initialization Time, s	N/A	2.7
Avg. Current, mA	N/A	735
Power Consumption, W	N/A	3.675
Power Efficiency, FPS/W	N/A	1.58

Tiny-YOLO was launched on RPi (see Fig.10) and on the bundle of RPi and Movidius NCS (see Fig.11). Table II shows that in the second experiment the performance is more than 6 times higher with the same power consumption.

From a large set of different MobileNet v1 variations, for the investigation of power consumption, we chose one with the depth 1.0 and input size 224 pixels. Power consumption and performance were measured for RPi (see Fig.12), Raspberry + Movidius NCS with an ammeter in RPi power circuit (see Fig.13) and with an ammeter in USB hub power circuit Fig.14. The last method was used also with two Movidius NCSs operating together at a time (see Fig.15).

Experimental setup with the ammeter in the USB hub power circuit (see Fig.2) allows us to study Movidius NCS power consumption without the influence of RPi consumption.

In Table III we can see that Movidius NCS increases productivity approximately 4.5 times with a slight increase in current consumption. At the same time, the pair of two NCSs, which increases the performance on desktop PC running the

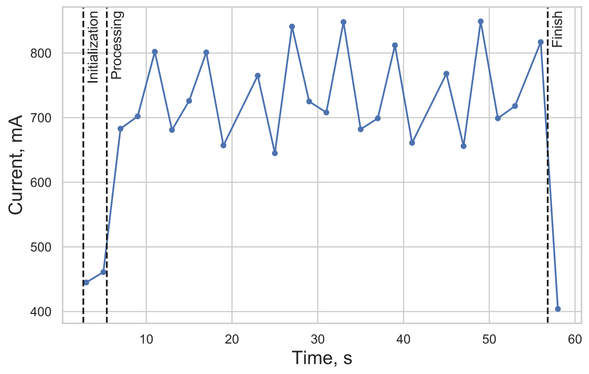

Fig. 9. Inception v1 on Raspberry Pi + NCS.

TABLE II
TINY-YOLO V2.

	RPi	**RPi + NCS**
FPS	0.40	2.57
Initialization Time, s	24.5	4.3
Avg. Current, mA	742	742
Power Consumption, W	3.710	3.710
Power Efficiency, FPS/W	0.11	0.69

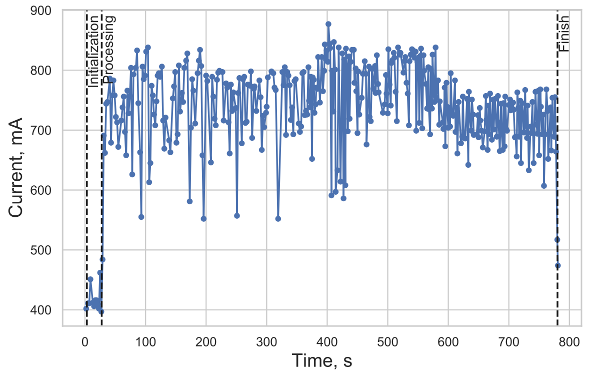

Fig. 10. Tiny-YOLO v2 on Raspberry Pi.

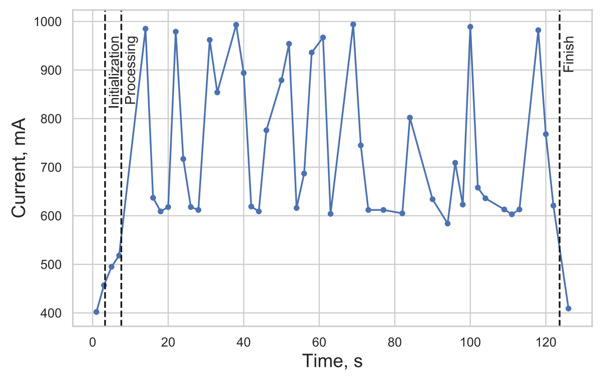

Fig. 11. Tiny-YOLO v2 on Raspberry Pi + NCS.

same test program with the same camera and USB hub (Table IV), does not guarantee a boost on RPi.

The source code for several Movidius NCSs test program is available in [14].

TABLE III
MOBILENET V1 1.0 224

	RPi	RPi + NCS	RPi + 2NCS
FPS	1.67	7.47	7.47
Initialization Time, s	163.5	3.2	4.4
Avg. Current, mA	639	646	-
Avg. Current (MO), mA	-	340	460
Power Consumption, W	3.195	3.230	-
Power Consumption (MO), W	-	1.7	2.3
Power Efficiency, FPS/W	0.44	2.35	-
Power Efficiency(MO), FPS/W	-	4.4	3.2

Long initialization time in case of single RPi experiment is caused by the need to load a large number of parameters (about 5 million) into the RAM.

TABLE IV
MOBILENET V1 1.0 224, INTEL CORE I7-8650U, UBUNTU 16.04

	PC + NCS	PC + 2 NCS
FPS	20	28

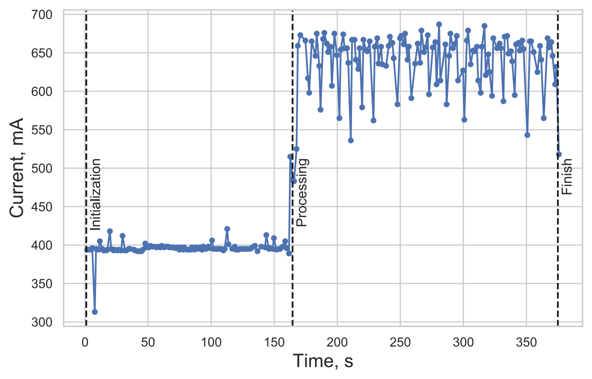

Fig. 12. MobileNet v1 on raw Raspberry

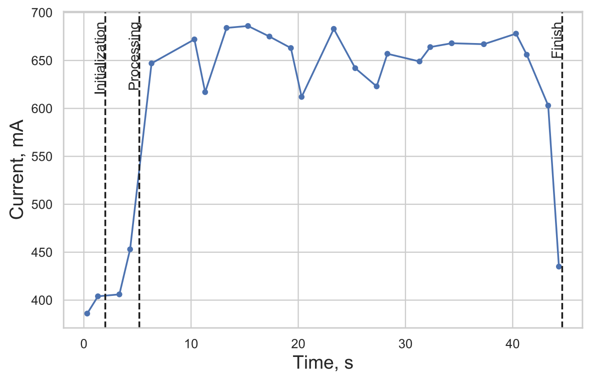

Fig. 13. MobileNet v1 on Raspberry + NCS

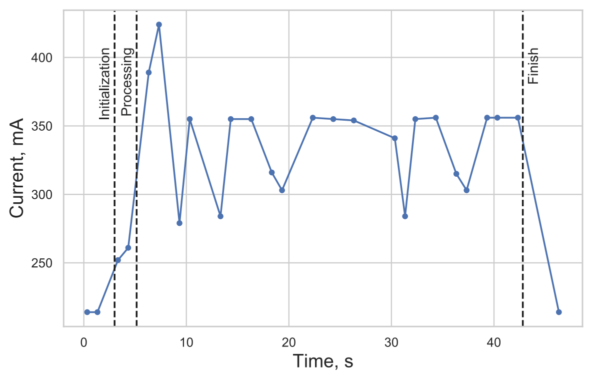

Fig. 14. MobileNet v1 on Raspberry + NCS, current through hub

IV. CONCLUSION

In this work, we have carried out an experimental assessment on how to efficiently use single embedded system and the embedded system empowered with an external GPU.

To successfully run the neural networks on the embedded systems we should monitor CPU Load and CPU Temperature, otherwise, the device will malfunction. In the case of

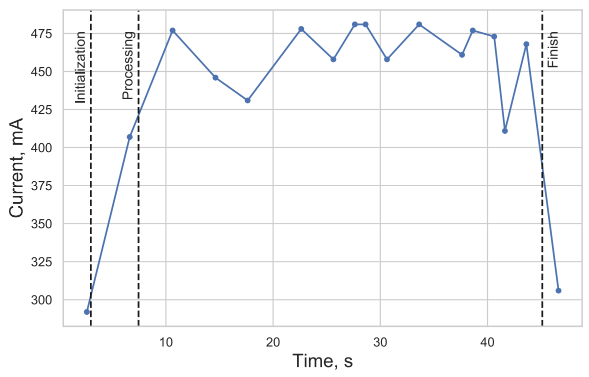

Fig. 15. MobileNet v1 on Raspberry + 2 NCS, current through hub

Raspberry Pi, we can launch most of MobileNet v1 models without such a risk. There are cases when an algorithm takes up too much space and its execution results in overheating when using heavier models. In this case we should use external GPU such as Movidius NCS for increasing the performance. Also, the set of Raspberry Pi and Movidius NCS is several times more effective than single Raspberry Pi in terms of power consumption. Another point is that when using multiple Movidius NCSs simultaneously with Raspberry Pi, unlike the PC, may not give the gain in performance.

The results of this work open up opportunities for intelligent and autonomous IoT deployments.

References

[1] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497 – 1516, 2012.

[2] P. Vlacheas, R. Giaffreda, V. Stavroulaki, D. Kelaidonis, V. Foteinos, G. Poulios, P. Demestichas, A. Somov, A. R. Biswas, and K. Moessner, "Enabling smart cities through a cognitive management framework for the internet of things," *IEEE Communications Magazine*, vol. 51, pp. 102–111, June 2013.

[3] A. Somov, D. Shadrin, I. Fastovets, A. Nikitin, S. Matveev, I. Oseledets, and O. Hrinchuk, "Pervasive agriculture: Iot-enabled greenhouse for plant growth control," *IEEE Pervasive Computing*, vol. 17, pp. 65–75, Oct 2018.

[4] I. Minakov and R. Passerone, "Pases: An energy-aware design space exploration framework for wireless sensor networks," *Journal of Systems Architecture*, vol. 59, no. 8, pp. 626 – 642, 2013.

[5] F. Khaled, O. Ondel, and B. Allard, "Optimal energy harvesting from serially connected microbial fuel cells," *IEEE Transactions on Industrial Electronics*, vol. 62, pp. 3508–3515, June 2015.

[6] F. Kaup, P. Gottschling, and D. Hausheer, "Powerpi: Measuring and modeling the power consumption of the raspberry pi," in *Local Computer Networks (LCN), 2014 IEEE 39th Conference on*, pp. 236–243, IEEE, 2014.

[7] A. Somov, A. Karelin, A. Baranov, and S. Mironov, "Estimation of a gas mixture explosion risk by measuring the oxidation heat within a catalytic sensor," *IEEE Transactions on Industrial Electronics*, vol. 64, pp. 9691–9698, Dec 2017.

[8] N. D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi, and F. Kawsar, "Squeezing deep learning into mobile and embedded devices," *IEEE Pervasive Computing*, vol. 16, no. 3, pp. 82–88, 2017.

[9] J. Chauhan, S. Seneviratne, Y. Hu, A. Misra, A. Seneviratne, and Y. Lee, "Breathing-based authentication on resource-constrained iot devices using recurrent neural networks," *Computer*, vol. 51, pp. 60–67, May 2018.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[11] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," *arXiv preprint*, 2017.

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[13] N. Melentev, "Complete guide to launch object detection of Raspberry Pi with Movidius." https://github.com/ndmel/tensorflow_object_detection_on_raspberry_pi, 2019.

[14] V. Prutyanov, "Experiments with Intel Movidius NCS for real-time camera image processing." https://github.com/viktor-prutyanov/movidius-webcam, 2019.