Classifying Smart IoT Devices for Running Machine Learning Algorithms

Aluizio Rocha Neto[1], Bárbara Soares[2], Felipe Barbalho[1], Luis Santos[1], Thais Batista[1] Flávia C. Delicato[3], Paulo F. Pires[3]

[1]Departamento de Informática e Matemática Aplicada (DIMAp) – Universidade Federal do Rio Grande do Norte, (UFRN), Natal – RN – Brazil

[2]Departamento de Engenharia de Computação e Automação (DCA) – Universidade Federal do Rio Grande do Norte, (UFRN), Natal – RN – Brazil

[3]Departamento de Ciências da Computação – Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro – RJ – Brazil

{aluiziorocha,barbaragabriellass}@gmail.com

{felipebarbalho.95,santosfluis19,thaisbatista}@gmail.com

{fdelicato,paulo.f.pires}@gmail.com

Abstract. *Tiny computers called System-on-a-Chip like Raspberry Pi have revolutionized the development of applications for Smart Home and Smart City. Some Machine Learning algorithms have been used to process a large amount of data produced by these Internet of Things (IoT) devices. An important issue in the context of processing IoT data is the decision on where the machine learning algorithm will run. To support this decision, it is necessary to classify the IoT devices according to their capabilities to run these algorithms, in terms of CPU performance, required memory, and energy demand. The aim of this paper is to classify IoT devices according to their capabilities to run machine learning algorithms, and reporting real experiments that validate the proposed classification.*

1. Introduction

The Internet of Things (IoT) paradigm [Atzori et al. 2010] has gained more and more notoriety in recent years. It encompasses an infrastructure of hardware, software, and services integrating objects from the physical world with the Internet. This scenario brings the opportunity for creating advanced applications, such as smart home, smart building and smart cities. These applications generally use several tiny and low-power sensors and networked devices, usually called System-on-a-Chip - SoC [Aitken et al. 2011]. Arduino[1] and Raspberry Pi[2] are two popular examples of these devices.

In the initial development of IoT, all data produced by any sensor should be transferred to servers normally hosted in Cloud Data Centers. This architecture places ever-increasing demands on communication and computational infrastructure with inevitable

[1]https://www.arduino.cc/

[2]https://raspberrypi.org

adverse effect on Quality-of-Service and Experience, given that over 25 billion devices are estimated to be added to the Internet by 2020, excluding PCs, tablets, and smartphones[3]. To address this challenge, the paradigms of Fog [Dastjerdi and Buyya 2016] and Edge Computing [Varghese et al. 2016] have emerged to allow the edge and end-user devices to be more Cloud independent. The main idea in these models is to process the data close to where it is generated, reducing the traffic and bandwidth consumption of the network. For instance, images from cameras capturing the objects and movements of what is happening in a variety of scenarios are very common as the main data for modern smart applications, such as information about vehicle traffic, crowded spaces or flood level of the river crossing the city, for example. To prevent sending all images from all cameras to a Cloud Data Center to process these datasets and to provide a fast response, this image processing should run on the embedded IoT devices.

However, image processing can only be done if the IoT image device has Computer Vision capabilities [Sebe et al. 2005], a method of Artificial Intelligence (AI) for acquiring, processing, analyzing and understanding digital images to extract high-dimensional data from the real world in order to produce numerical or symbolic information, i.e., models. The AI algorithms used in these analysis are in constant evolution to improve their accuracy. Machine Learning (ML) techniques have been used for automating the data model acquisition and updating processes, adapting task parameters and representations, and using experience for generating, verifying and modifying hypotheses about theses models [Sebe et al. 2005]. Traditionally all Machine Learning algorithms run on powerful computers due to high demand for computation and memory requirements. In the context of IoT, the data is produced by devices located at the edge of the Cloud that have limited hardware resources. In order to balance processing with network traffic to the Cloud a question that arises is the decision of where the machine learning algorithm will run. Some researches have shown that if the IoT device have the ability to extract and send to the Cloud only the information learned, this reduces considerably the network demand and improves the responsiveness of the systems.

However, to run these learning algorithms it is necessary to classify the IoT devices according to their capabilities to run them, in terms of CPU performance, required memory, and energy demand. As far as we are concerned, the literature does not present such a classification. The goal of this paper is twofold: (i) to classify the IoT devices according to their capabilities to run machine learning algorithms. (ii) to report real experiments that validate the proposed classification. As contribution, we hope to provide useful information to guide the decision making of IoT application designers/developers. The remainder of the paper is organized as follows. In section 2 we present a background review of recent research works related to Machine Learning and Edge Computing. Section 3 presents the proposed classification for the ”smart” IoT devices. Section 4 shows some experiments we conducted using the IoT devices to evaluate the performance of ML algorithms in such devices. Finally, section 5 brings our final remarks.

2. Background review

Machine learning is a research field that formally focuses on the theory, performance and properties of learning systems and algorithms. It is a highly interdisciplinary field build-

[3]http://www.gartner.com/newsroom/id/2636073

Table 1. Comparison of machine learning technologies [Qiu et al. 2016].

Learning types	Data processing tasks	Distinction norm	Learning algorithms
Supervised learning	Classification / Regression / Estimation	Computational classifiers	Support Vector Machine
		Statistical classifiers	Naive Bayes Hidden Markov model Bayseian networks
		Connectionist classifiers	Neural networks
Unsupervised learning	Clustering / Prediction	Parametric	K-means Gaussian mixture model
		Nonparametric	Dirichlet proc. mix. model X-means
Reinforcement learning	Decision-making	Model-free	Q-learning R-learning
		Model-based	TD learning Sarsa learning

ing upon ideas from many different kinds of fields such as artificial intelligence, optimization theory, information theory, statistics, cognitive science and many other disciplines of science [Qiu et al. 2016].

Generally, the field of machine learning is divided into three subdomains: supervised learning, unsupervised learning, and reinforcement learning. Briefly, supervised learning requires training with labeled data which has inputs and desired outputs. Unsupervised learning does not require labeled training data and the environment only provides inputs without desired targets. Reinforcement learning enables learning from feedback received through interactions with an external environment [Qiu et al. 2016]. Table 1 presents these three machine learning techniques from different perspectives.

Supervised and unsupervised learning mainly focus on data analysis while reinforcement learning is preferred for decision-make problems. Most traditional ML based systems are designed with the assumption that all the collected data would be completely loaded into memory to start processing. Nowadays, there is a great need to develop efficient and intelligent learning methods to cope with future data processing demands [Qiu et al. 2016].

Deep Learning (DL) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. DL enables computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains [LeCun et al. 2015].

Currently, deep learning algorithms are not widely used on IoT devices because they are often very resource consuming for the system (e.g., memory, computing and power). In [Lane et al. 2015a], the authors present a measurement study of running common deep learning models (AlexNet [Krizhevsky et al. 2012], SVHN [Netzer et al. 2011], DeepKWS [Chen et al. 2014], and DeepEar [Lane et al. 2015b]) to process audio and image sensor data on representative mobile and embedded platforms:

- Qualcomm Snapdragon 800 is widely used in smartphones and tablets (3 processors: a Krait 4-core 2.3 GHz CPU, an Adreno 330 GPU and a 680 MHz Hexagon

Table 2. Execution time (msec) for running DL algorithms in three hardware platforms [Lane et al. 2015a].

	Type	Size (bytes)	Tegra		Snapdragon		Edison
			CPU	GPU	CPU	DSP	CPU
Deep KWS	DNN	241K	0.8	1.1	7.1	7.0	63.1
DeepEAR	DNN	2.3M	6.7	3.2	71.2	379.2	109.0
SVHN	CNN	313K	15.1	2.8	1,615.5	-	3,352.3
AlexNet	CNN	60.9M	600.2	49.1	159,383.1	-	283,038.6

DSP, 1GB of RAM).

- Intel Edison, the smallest and least computational powerful of all tested hardware (500MHz dual-core Atom "Silvermont" CPU assisted by a 100 MHz Quark processor, 1 GB of RAM).
- Nvidia Tegra K1 developed for extreme GPU performance in IoT context (Kepler 192-core GPU, 2.3 GHz 4-core Cortex CPU, 2GB of RAM).

Table 2 shows the times in milliseconds the authors have obtained for running two Deep Neural Networks (DNN) - Deep KWS and DeepEar, and two Convolutional Neural Networks(CNN) - SVHN and AlexNet, on theses devices. They found that almost all DL model and processor combinations can run on such platforms. Even large-scale models like AlexNet (60.9MB size) are supported by the weakest of the processors they used (an Intel Edison). This result suggests that DL models of similar architecture targeting different inference tasks will also function to some degree. They conclude that the range of inference tasks offered by this set of DL models is enormous, and comprise tasks generally not seen on IoT hardware.

[Kamath et al. 2016] introduced EdgeSGD, a decentralized Stochastic gradient descent algorithm suitable for machine learning and analytics on the edge of the network. This method can be applied to a wide range of problems arising in decentralized machine learning. In the paper, they used the proposed algorithm to learning/predicting seismic anomalies via real-time imaging and evaluated the performance of the algorithm on an edge computing testbed - a cluster composed of 16 BeagleBone Black[4] (CPU single-core 1GHz, 512MB of RAM). They also compared the proposed solution with other existing distributed computation methods such as MapReduce, DGD and EXTRA, and examined in particular the effects of node/link failure and communication cost. Such a comparative analysis showed that edge processing with EdgeSGD is quite feasible.

In recent years, a significant amount of research has been done using the Raspberry Pi as the edge node to process IoT data. [Anandhalli and Baligar 2017] proposes a video processing algorithm that detects, tracks and counts vehicles on a road, applying a Kalman filter to track vehicles, and applying individual vehicle detection through the HSV (Hue, Saturation, Value) color spectrum. The algorithm runs on a Raspberry Pi 3 (1.2GHz quad-core ARMv8, 1GB of RAM) with an embedded camera and uses the OpenCV[5] (Open Source Computer Vision Library) that contains a set of filters for image processing. The work also compares the performance of running the same algorithm on a

[4]https://beagleboard.org/black

[5]https://opencv.org/

desktop PC (2.5 GHz dual-core, 4GB of RAM) and it took 72.7 ms to process one frame whereas Raspberry Pi needed 81.589 ms, only 9 ms slower.

In [Sajjad et al. 2017] the authors developed a face recognition framework for law-enforcement services in smart cities. This framework uses a small-sized portable wireless camera mounted on a police officer's uniform to capture a video stream, which is passed to a Raspberry Pi 3 in the officer's car for face detection and recognition. To accomplish the facial recognition, they use initially the Viola-Jones face detection algorithm to find all the faces present in the live video stream sent by the camera. Then, the ORB (Oriented FAST and Rotated BRIEF) method is executed to extract from the identified faces all their features which are transmitted to a trained support vector machine classifier in the cloud. Instead of sending the entire captured image, only the features extracted from the face are sent, thus saving transmission power and bandwidth.

An issue that came up in reading these articles was what criteria were applied to choose the IoT devices used in the experiments to process data at the edge of network. What hardware specifications were important for running the ML algorithms. This paper analyses these aspects, proposes a classification for choosing an IoT device to run ML algorithms, and reports some experiments using IoT devices to learn with its own data.

3. Smart IoT Device Classification

The ability to process and understand the IoT data to obtain some higher level information is fundamental for the Machine Learning techniques. From a basic statistic, like the average temperature in a room, to a complex information, such as "how many people are in the room right now?" all these informations require the data analysis for extracting useful features and representations of the data. These kinds of data processing requires different levels of hardware resources to be done. Thus, we proposed an IoT device classification so that we can match the algorithm resources demand with the hardware class of the most popular IoT devices found on the market.

In order to perform this type of data analysis, the device must have the ability to fuse (combine) all the data and extract the useful information. For example, a suddenly temperature of 16 degree when the average temperature is 24 degree can be interpreted as a data noise and it must be discarded. According to [Nakamura et al. 2007], *Information Fusion* deals with three levels of data abstraction: measurement, feature, and decision, and it can be classified into four categories:

- *Low-Level Fusion*. Also referred to as *signal (measurement) level fusion*. Raw data are provided as inputs, combined into new piece of data that is more accurate (reduced noise) than the individual inputs.
- *Medium-Level Fusion*. Attributes or features of an entity (e.g., shape, texture, position) are fused to obtain a feature map that may be used for other tasks (e.g., segmentation or detection of an object). This type of fusion is also known as *feature/attribute level fusion*.
- *High-Level Fusion*. Also known as *symbol* or *decision level fusion*. It takes decisions or symbolic representations as input and combines them to obtain a more confident and/or a global decision.
- *Multilevel Fusion*. When the fusion process encompasses data of different abstraction levels — when both input and output of fusion can be of any level (e.g.,

Table 3. Classification of Smart IoT devices according to their capacities.

Class	Hardware capacity	Power consumption	Suitable algorithms	Main application
1	No storage, low CPU and memory	$\leq$ 1W	Basic computation	Data generation
2	storage $\leq$ 4GB, memory $\leq$ 512MB CPU single-core	$\leq$ 2W	Basic statistic	Low-level data fusion
3	storage $\leq$ 8GB memory $\leq$ 2GB CPU quad-core	$\leq$ 4W	Classification / Regression / Estimation	Mid-level data fusion
4	storage $\geq$ 16GB memory $\geq$ 4GB CPU and GPU	$\leq$ 8W	Prediction / Decision-making	High-level data fusion
5	High	High	Any	Autonomous system

a measurement is fused with a feature to provide a decision) — multilevel fusion takes place.

Since any device connected to the Internet can generate and process data to some degree, it is important to define a classification for such devices so that we can identify which ML algorithms can be deployed to each equipment model. Usually, ML algorithms have to process a huge amount of data, requiring powerful CPU and large memory. These kind of resources are very limited in the majority of IoT devices. In addition, intense CPU usage has a significant use of power which is also limited in IoT devices.

Table 3 shows the proposed Smart IoT Device Classification. The hardware capacity and power consumption data for each class are based on related researches and our experiments using some IoT devices running intelligent solutions. This classification considers only IoT devices for applications used in smart cities, buildings or home, i.e. we are not considering hardwares used in mobile devices, such as phones or tablets.

The main purpose of class 1 devices is to collect data from their sensors where the dataset is small, analogical and continuous, such as sensors measuring temperature, humidity, presence, noise, gases, etc. This kind of device also have some actuators such as LEDs, relay module and servo motors to do some actions in the ambient whenever it is needed. Class 2 devices are a step further in the ability of collecting, processing, and storing data. Unlike class 1 devices that have microcontrollers, they have true CPU, RAM memory and an operating system to control the whole system. But, due to their limited resources they can only process some basic statistics to produce the low-level data fusion.

Devices class 3 and 4 are the ones really used for running Machine Learning algorithms on the edge. The main difference between them is that class 4 devices have a powerful GPU for parallel processing, which is very important in the execution of algorithms of training and using neural networks. Class 3 device can use a trained neural network to make the medium-level fusion of the data and extracting the relevant features of the context, like a detection of an object. Class 4 device can go further and taking

Table 4. Examples of IoT devices and their classifications.

Class	Device	CPU	GPU	Memory / Storage	Power Consump.
1	Arduino Mega	Microcontroller ATmega 8-bit 16 MHz	None	32 KB / None	$\leq$ 408mW
	NodeMCU ESP-12	Microcontroller ESP8266 32-bit 80 MHz	None	80 KB / None	$\leq$ 561mW
2	Raspberry Pi Zero	ARM1176 single-core 1 GHz	Broadcom VideoCore IV	512 MB / MicroSD card	0.5W - 1.2W
	Beaglebone Black	ARM Cortex-A8 single-core 1 GHz	PowerVR SGX530	512 MB / 4GB + card	1.1W - 2.15W
3	Raspberry Pi 3	ARM Cortex-A53 quad-core 1.2 GHz	Broadcom VideoCore IV	1 GB / MicroSD card	1.2W - 3.7W
	ODROID XU4	ARM Cortex-A15 octa-core 2 GHz	ARM Mali-T628	2 GB / MicroSD card	2.1W - 13.2W
4	NVIDIA Jetson TX2	ARM Cortex-A57 quad-core 2 GHz + NVIDIA Denver2 dual-core 2GHz	NVIDIA Pascal 256 CUDA cores	8 GB / 32GB + SATA	7.5W - 15W

decisions using a set of higher level data fusion. It can also retrain the neural network for running into a class 3 device.

A class 5 is a multilevel-fusion device and can be considered as an autonomous system, which can act automatically to prevent a situation in the scenario. The best example of such a device is the computational system used into self-driven vehicles. These systems are expected to operate flawlessly irrespective of weather conditions, visibility, or road surface quality and have the capacity to deal with terabytes of datasets produced by their sensors[6].

Table 4 shows the instantiation of classification applied to some IoT devices found in the market. The power consumption range was obtained from the manufacturer specifications and forums of users on the Internet. Most of the IoT devices use a MicroSD card slot as a storage unit for their data, and the BeagleBone Black comes with an internal 4GB flash memory.

4. Probing the Smart IoT Device Classification

To validate our proposed classification, this section describes some of our experiments that we have developed using IoT devices to make systems more smarter. The main characteristic of these experiments is the application of edge computing paradigm, i.e., the processing of the sensors data in the device that produces these data, reducing network delays and bandwidth.

4.1. Smart Pole Project

The first solution is related to out Smart Campus project where we have developed a Smart Pole powered by solar panel that acts as a sensing and connectivity infrastructure for smart cities. This solar city pole is carried with Wi-Fi radios, IP camera, and an

[6]https://devblogs.nvidia.com/training-self-driving-vehicles-challenge-scale/

IoT device and some of the services it can offer to the citizens are: Internet Wi-Fi Hot-spot, sensing of environmental pollution, spot for an automated parking space availability system, autonomous and intelligent lighting spot, and place for digital panels with useful information updated in real time.

The main issue of any solar powered system is the battery setup and how the embedded equipment will use the energy produced by the solar panel and stored into battery for the usage at night or raining days. We have been using the solar charger controller EPsolar VS2024BN[7] to manage the energy in the pole. Its main function is to limit the rate at which electric current is added to or drawn from battery and it cuts off electric current from battery if its voltage reaches an under boundary. The connectivity and energy for Wi-Fi radios and IP camera is provided by a Power over Ethernet (PoE[8]) Switch. Figure 1 shows a picture of the pole and the schema of the energy and network topologies.

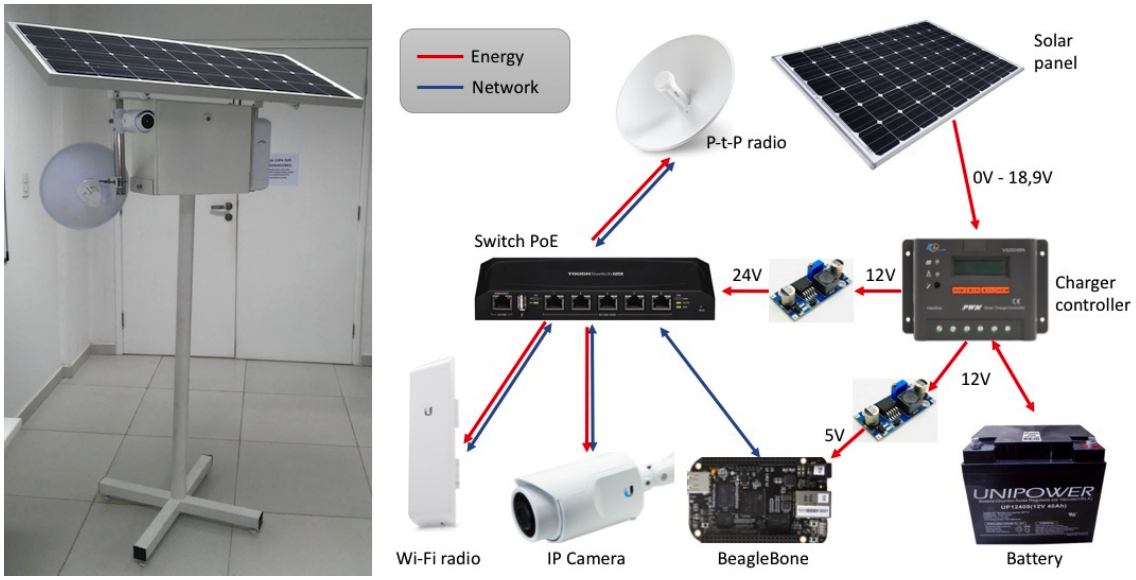

Figure 1. The Smart Pole and its energy and network topologies.

The IoT device Beaglebone Black used in the pole is the sensing platform and the data collector and analyzer for the energy produced and consumed in the system. As a class 2 device, it does the low-level fusion of sensors data and can compute several statistics about the pole operation and datasets. Every 5 minutes the Beaglebone asks the charger controller via an RS-485 interface all data about the energy management in the system. All data are stored on the device in a SQLite[9] database and after one year this database has approximately 100 thousand records that occupies only 12MBytes of the BeagleBone's 4GB internal storage flash memory. By mining this dataset the BeagleBone can predict a critical situation when the battery charge is getting too low and the charger controller will cut off all the energy that powers the equipments. Before this situation happens, the BeagleBone can act by sending SNMP[10] set commands to the switch to

[7] http://www.epsolarpv.com/en/index.php/Product/pro_content/id/166/am_id/136
[8] https://en.wikipedia.org/wiki/Power_over_Ethernet
[9] https://www.sqlite.org/
[10] Simple Network Management Protocol (http://www.snmp.com/protocol/)

Table 5. Running a complex SQL query in different software and hardware platforms

	SQL query response time (msec)
MySQL @ Cloud server	2,280
SQLite @ Cloud server	291
SQLite @ Beaglebone	3,113
SQLite @ Raspberry Pi 2	3,143

disable the port(s) connected the IP camera and/or the Wi-Fi radio to save energy until the solar panel can produce power again.

To compare the performance of the BeagleBone running SQL queries that use statistical functions (COUNT, MAX, MIN, and SUM) we have exported this database to a MySQL server (Intel Xeon quad-core 2.6GHz and 4GB of RAM) in the Cloud to get its response time as reference. Table 5 shows the results for running the SQL query that gets all the power cuts and subsequent period of inactivity due to low power in the battery. This query has 3 nested subqueries and applies the MAX function to a 'timestamp' type column of the table. Surprisingly the response time of MySQL running on the Cloud server was close to the time of SQLite running on BeagleBone and Raspberry Pi 2. This suggests that any ML algorithm that needs statistics to learn using large datasets can be done in the edge using a Smart IoT device class 2.

4.2. Smart Place Project

The second solution is also related to our Smart Campus project and it is a system named Smart Place which uses sensors, camera, and Raspberry Pi as the IoT device manager to intelligently control the air conditioners (AC) in smart buildings. This system was developed aiming to contribute to energy saving by automatically managing the usage of AC at the Federal University of Rio Grande do Norte.

The Smart Place system detects the presence of people in the room and automatically turns on the air conditioner and then turns off if nobody has been detected in the last 15 minutes. It also checks the room reservation data for the next minutes before turning the air conditioner off, so that no AC will have its status switched (on-off-on) in a short period of time, which would increase the consumption of energy instead of contributing to any savings.

Initially, only a motion sensor was used to detect the presence of people in the room. But due to its distance limitation, specially in environments that are larger like a classroom, an embedded Raspberry Pi's camera was integrated to improve the efficiency of the system. Figure 2 shows an diagram of the Smart Place system in a room and a schema of the device, sensors, and actuator (infrared LED) used. The Raspberry Pi commands the AC sending infrared signals to its IR receiver, like an AC remote control. All data collected from the sensors and camera (temperature, humidity and presence) is stored in a SQLite database and processed in the Raspberry Pi to make the decision of turning on or off the AC. For monitoring the whole system in all rooms the data is also sent to a Cloud server.

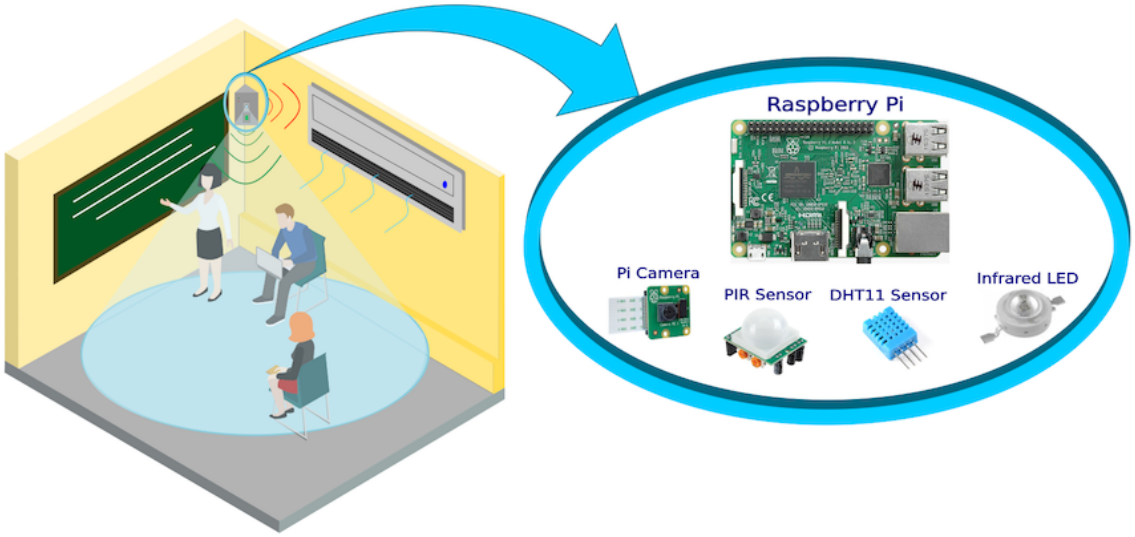

Figure 2. The Smart Place system.

Table 6. Running CNN MobileNet in different computers

	DL algorithm response time (msec)
Notebook	130
Raspberry Pi 3	1,609
Raspberry Pi 2	2,398
Raspberry Pi Zero	16,383

With the image of the room captured with the Raspberry Pi's camera, classification, counting, and localization of people in such place was possible. In order to do theses processes, we have been using object detection through Deep Learning techniques available in the OpenCV library. After testing several methods for object detection, Single Shot Detectors (SSDs) [Liu et al. 2016] was the method that better suit our system, being even faster and more precise than the YOLO method [Redmon et al. 2016], and by combining it with the CNN model MobileNet [Howard et al. 2017], we got an efficient and fast deep learning method for object detection which can be used in any resource constrained device, like the Raspberry Pi. Table 6 shows our experiments using a notebook with webcam (MacBook Air Intel dual-core i5 1.3GHz, 4GB of RAM) and three versions of Raspberry Pi to run the CNN MobileNet model in order to detect and count the people in the room.

The response time of the Raspberry Pi 3 and 2 (class 3 devices) is quite interesting. Both have quad-core processors and 1GB of RAM and the difference of Pi 3 time for the notebook time was about 1.5 seconds slower. For this reason and prospecting the potential of such devices in the development of smart applications, the academic community has been researching more computationally efficient neural networks and with a smaller memory/processing requirement. One of such initiative is the SqueezeNet [Iandola et al. 2016].

5. Final Remarks

The Internet of Things technology is being increasingly used mainly with the advent of smart solutions (smart city, smart building, home automation, etc.). Due to the rapid evolution of System-on-a-Chip, there is a big marathon for developing intelligent systems where small devices process their own data and take decisions to improve user experiences. This novel paradigm, known as Edge or Fog Computing, will allow the Internet to grow even more, improve the user experience and reduce the dependency on Cloud infrastructures.

Analyzing the machine learning techniques and how the ML algorithms run on resource constrained devices, this paper proposed a Smart IoT Device Classification. This classification helps developers choosing the IoT device that better matches the hardware resources with the data processing and learning requirements for their smart systems. Some experiments of using the IoT devices discussed in this paper were also presented, showing the potential this tiny devices have in smart solutions.

We believe that machine learning algorithms will also evolve along with IoT devices because there is a whole new market for intelligent systems using the Internet as the main infrastructure. This market is pushing forward the development of new hardware, software, and network protocols to address all the challenges present in such systems.

References

Aitken, R., Flautner, K., and Goodacre, J. (2011). High-performance multiprocessor system on chip: Trends in chip architecture for the mass market. In Hübner, M. and Becker, J., editors, *Multiprocessor System-on-Chip - Hardware Design and Tool Integration*, pages 223–239. Springer.

Anandhalli, M. and Baligar, V. P. (2017). A novel approach in real-time vehicle detection and tracking using raspberry pi. *Alexandria Engineering Journal*.

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, 54(15):2787–2805.

Chen, G., Parada, C., and Heigold, G. (2014). Small-footprint keyword spotting using deep neural networks. In *ICASSP*, pages 4087–4091. IEEE.

Dastjerdi, A. V. and Buyya, R. (2016). Fog computing: Helping the internet of things realize its potential. *IEEE Computer*, 49(8):112–116.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *Computer Vision and Pattern Recognition (cs.CV)*.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <05mb model size.

Kamath, G., Agnihotri, P., Valero, M., Sarker, K., and Song, W.-Z. (2016). Pushing analytics to the edge. In *GLOBECOM*, pages 1–6. IEEE.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Bartlett, P. L., Pereira, F. C. N., Burges, C. J. C.,

Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114.

Lane, N. D., Bhattacharya, S., Georgiev, P., Forlivesi, C., and Kawsar, F. (2015a). An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices. In Xu, C., Zhang, P., and Sigg, S., editors, *Proceedings of the 2015 International Workshop on Internet of Things towards Applications, IoT-App 2015, Seoul, South Korea, November 1, 2015*, pages 7–12. ACM.

Lane, N. D., Georgiev, P., and Qendro, L. (2015b). Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In Mase, K., Langheinrich, M., Gatica-Perez, D., Gellersen, H., Choudhury, T., and Yatani, K., editors, *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp 2015, Osaka, Japan, September 7-11, 2015*, pages 283–294. ACM.

LeCun, Y., Bengio, Y., and Hinton, G. E. (2015). Deep learning. *Nature*, 521(7553):436–444.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Scott Reed4, C.-Y. F., and Berg, A. C. (2016). Ssd: Single shot multibox detector. *Computer Vision – ECCV 2016*, 9905:21–37.

Nakamura, E. F., Loureiro, A. A. F., and Frery, A. C. (2007). Information fusion for wireless sensor networks: Methods, models, and classifications. *ACM Comput. Surv.*, 39(3).

Netzer, Y., Wang, T., Coates, A., Bissacco, R., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning.

Qiu, J., Wu, Q., Ding, G., Xu, Y., and Feng, S. (2016). A survey of machine learning for big data processing. *EURASIP J. Adv. Sig. Proc.*, 2016:67.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788.

Sajjad, M., Nasir, M., Muhammad, K., Khan, S., Jan, Z., Sangaiah, A. K., Elhoseny, M., and Baik, S. W. (2017). Raspberry pi assisted face recognition framework for enhanced law-enforcement services in smart cities. *Future Generation Computer Systems*, November 2017.

Sebe, N., Cohen, I., Garg, A., and Huang, T. S. (2005). *Machine Learning in Computer Vision*, volume 29 of *Computational Imaging and Vision*. Springer.

Varghese, B., Wang, N., Barbhuiya, S., Kilpatrick, P., and Nikolopoulos, D. S. (2016). Challenges and opportunities in edge computing. In *2016 IEEE International Conference on Smart Cloud, SmartCloud 2016, New York, NY, USA, November 18-20, 2016*, pages 20–26.