Computing manuscript No.
(will be inserted by the editor)

Smart Connected Parking Lots based on Secured Multimedia IoT Devices

Mohammed Amine Merzoug · Ahmed Mostefaoui · Gabriele Gianini · Ernesto Damiani

Received: June 2020 / Accepted: date.

Abstract In this paper, we present a smart connected parking lots solution to automatically count and notify drivers about empty parking spots in major cities. As its name implies, the proposed smart IoT system has two operating phases: (i) continuous counting of empty spots in the monitored far-apart parking lots, and (ii) instantaneous driver notification through a lightweight MQTT mechanism. This notification system relies only on information collected from the pre-installed multimedia devices (no other apparatus installation or maintenance such as ground sensors is required). To validate the proper operation of our solution, we have implemented a small-scale version of it and assessed its performance while considering different classical and lightweight deep learning techniques (MobileNetV2, ResNet-50, YOLOv3, SSD-MobileNetV2, Tiny-YOLO, SqueezeDet, and SqueezeDet pruned with ℓ_1-*norm*). The experiments have confirmed the proper operation, efficiency, ease of deployment, and ease of extension of our system. They also confirmed that lightweight deep learning solutions are more adequate for small-sized resource-constrained embedded systems. They are more efficient in terms of inference time, size, resource consumption, and yield an accuracy that is close to that of classical solutions.

Keywords In-city parking · parking spot availability · deep learning · connected parking lots · Internet of things.

Mohammed A. Merzoug
Computer Science Dept., University of Batna 2, Batna 05078, Algeria.
E-mail: amine.merzoug@univ-batna2.dz

Ahmed Mostefaoui
University of Burgundy Franche-Comté, France. E-mail: ahmed.mostefaoui@univ-fcomte.fr

Gabriele Gianini
EBTIC/Khalifa University of Science and Technology and Università degli Studi di Milano, Italy. E-mail: gabriele.gianini@unimi.it

Ernesto Damiani
Khalifa University of Science and Technology and Università degli Studi di Milano, Italy.
E-mail: ernesto.damiani@unimi.it

1 Introduction

Finding a parking spot in a densely populated city is usually a tedious time-consuming task, especially during rush hours. According to numerous recent and old studies [39], drivers travel each year several extra kilometers to find empty parking spaces, thus, wasting energy, time and unintentionally contributing to environmental pollution. To remedy these issues while improving vehicle services and offering a comfortable mobility experience, several solutions have been proposed (Section 5 will present some of these works). In fact, even though the in-city vehicle parking issue is not recent, it is still however open and challenging. The approach presented in this paper partitions this problem into two sub-problems: *parking smartness* and *parking connectivity*. The first issue is related to how can one make a parking lot smart and autonomous? In other words, how can an IoT system efficiently and reliably distinguish regions occupied by vehicles from empty spots? The second one is linked to the most appropriate communication paradigm used to inform, in real-time, drivers about parking spot availability in any zone of their choice?

We present in this paper a smart IoT (Internet of Things) notification system that addresses these issues. Our proposal is based on the use of small-sized, cheap multimedia devices able to capture images and process them locally. The use of such devices is primarily motivated by their low prices (economically efficient to cover a large city) and their deployment easiness. We hence report about the rationale of its design and its implementation: we assessed the system and verified that it is suitable for the setting for which it was tailored (i.e; it can efficiently monitor and manage parking lots that are scattered across large densely-populated cities). The system, in addition to its deployment easiness, can also be easily extended by adding/connecting new parking lots at any given moment.

The operation of the proposed system is divided into two main complementary phases: (1) periodic deep learning-based image processing and (2) instantaneous IoT communication. First, based on image sensing and using efficient lightweight deep learning techniques (to cope with the performance limitation of IoT devices), this system continuously and automatically detects and counts the empty places in a whole city or any designated (connected) parking lot. Then, using efficient lightweight IoT communication, the system instantaneously notifies the subscribed drivers about the availability of parking spots in any covered region of their interest.

To assess its performance, a small-scale version of our proposed system was implemented, deployed, and evaluated, and different deep learning solutions (lightweight and conventional ones) were considered. The obtained experimental results validated the smooth operation and effectiveness of our system in terms of real-time notifications. The conducted experiments have also confirmed the ease of deployment and ease of extension and adaptability of this system. Indeed, the proposed solution is characterized by not only its ease of deployment but also its low cost (especially when compared with other hardware approaches that rely on ground sensors and devices that must be placed

on every parking spot). In our system, instead of a huge number of devices and sensors, a single multimedia device with the appropriate field of view can be utilized to monitor an entire parking lot [7]. Hence, no additional costs are necessary to install and inevitably maintain the densely deployed devices, particularly in major large cities.

The remainder of this paper is organized into five sections as follows. Section 2 presents both the considered lightweight deep learning techniques and lightweight IoT notification protocol. Section 3 details the proposed connected parking lots solution. Section 4 provides an implementation example of the proposed system and also presents and comments on the obtained experimental results. Section 5 reviews some approaches related to parking-spots occupancy and parking-lots monitoring. Finally, Section 6 concludes the paper and discusses some future research directions.

2 Background

Since the operation of the proposed solution is based on resource-constrained devices, both lightweight smart computations and lightweight IoT communication are a must. These two essential mechanisms are presented in the following two subsections.

2.1 Lightweight deep learning

Deep learning, as more in general, artificial intelligence aims to endow machines with human capabilities (seeing, hearing, understanding, learning, etc.). Hence, allowing them to classify images, detect vehicles and pedestrians, recognize voices, understand natural language, etc. Most deep learning techniques use artificial neural network architectures to exploit the in-hand data and include a representation learning phase in which they discover which features are the most important, for instance for distinguishing each class of objects to be classified [13]. Deep learning algorithms, thus, typically build a model for feature extraction and a model for classification or regression.

For the purposes of the present paper, it is useful to distinguish two families of *lightweight* deep learning approaches: (i) the first consists of designing from scratch modern, less deep, and less complex architectures with fewer parameters, and (ii) the other consists of compressing/optimizing existing architectures, by using a number of different techniques. In what follows, we will briefly review these two families of approaches. We start, in Section 2.1.1, by presenting some techniques to optimize classical CNN architectures[1]. Then, in Section 2.1.2, we will present the most relevant new lightweight CNN architectures.

[1] CNN: Convolutional Neural Network.

2.1.1 Optimization of classical architectures

As neural networks are increasingly complex, larger, and highly structured, several techniques have been proposed to minimize their size after training. Among these solutions the most important are (a) the *pruning* [5][21][26] and (b) *arithmetic approximation/quantization* approaches (fixed-point quantization, low precision weights, low precision multiplications, etc.) [4][17][27]. In this paper, we focus on the pruning of neural networks, which is an old idea that can be summarized as follows [25]. Among the many parameters of a neural network, some are redundant and do not contribute much to the output. Moreover, as the number of parameters increases, the over-fitting problem can arise with devastating effects on the overall system performance. To remedy this issue, numerous pruning techniques have been proposed to reduce the network size while preserving its precision (e.g., selectively removing weights, etc.).

2.1.2 New lightweight architectures

In this section, we present different CNN architectures that have been successfully applied to classify and detect objects (such as vehicles, human faces, etc.) using embedded systems. When compared with more conventional deep learning approaches (such as AlexNet [23] and ResNet [19]), these lightweight architectures can be appreciated for their very low number of parameters, which allows them to reduce memory usage, inference time, energy consumption, and renders them very suitable for small devices with limited resources. The architectures presented here – namely MobileNet [20,36], SqueezeNet [22], SqueezeNext [15], and ShuffleNet [43] – follow different design strategies.

A) MobileNet [20,36]: the distinguishing feature of these networks, which are specifically tailored for mobile and resource-constrained embedded systems, is the replacement of conventional convolutions with new ones called *depth-wise separable convolutions*. A standard convolution layer convolves the input data with a sliding filter and combines the results into the output data. Depth-wise separable convolutions have an identical goal, but combine the mechanism of depth convolution and of separability, by a factorization which makes the computation more lightweight. Let us consider, for the sake of illustration, a case where the input data are three-dimensional (e.g., represent a spatial 2D image along with its three color channels depth), a conventional convolution would slide a 3D filter across the image to produce the output. A depth-wise convolution would separate the three depth channels into as many 2D convolutions, if it is a depth-wise separable convolution it would also factor each 2D filter into two 1D filters. Once the distinct filtering operations are completed, one can recompose the three resulting operations (logically resulting in a depth array associated with each pixel) by a depth-wise 1×1 convolution, i.e. a point-wise convolution. Thanks to the combination of those mechanisms, MobileNet architectures reduce significantly both the number of calculations and

the size of the model/code in comparison to architectures using the standard convolutions. The operational characteristics can be controlled by two devoted parameters, that allow to tune the reduction of number of operations:

- *Width multiplier parameter* ($\alpha \in [0, 1]$): aims to reduce the number of channels. Instead of producing n channels in each layer, it produces $\alpha \times n$. This multiplier can be utilized to effectively trade-off between the desired accuracy and performance.
- *Resolution multiplier parameter* (ρ): resizes the input image. This resolution multiplier can also be utilized to reduce the internal representation of each layer.

B) SqueezeNet [22]: this model, which is a small CNN architecture, achieves the same precision as that of its predecessor AlexNet [23], but with 50× fewer parameters. To achieve this performance, the following three strategies have been adopted:

- *Replacing the 3×3 filters with 1×1 filters*: since 1×1 filters have 9× fewer parameters than 3×3 filters, the majority of the applied filters must be 1×1.
- *Decreasing the number of input channels to the 3×3 filters*: to maintain a small total number of parameters, it is important not only to decrease the number of 3×3 filters, but also to decrease the number of input channels to these filters. SqueezeNet accomplishes this task by utilizing specific *squeeze* layers.
- *Delayed downsampling:* the delayed reduction (e.g., pooling layers, etc.) gives larger activation maps, which in turn lead to higher classification accuracy [18].

C) SqueezeNext [15]: is a new family of neural networks whose design was guided by previous light architectures such as the one described in [22]. These new networks are capable of reproducing the precision of AlexNet [23] with 112× fewer parameters. To do so, SqueezeNext makes the following four changes to the basic SqueezeNet architecture:

- To considerably reduce the total number of parameters used with the 3×3 convolutions, a more aggressive channel reduction is applied by incorporating a two-level squeeze module.
- To further reduce the size of the model, separable 3×3 convolutions are used and the additional 1×1 *expand* branch that comes after the squeeze module is removed.
- To allow the training of a much deeper network without the vanishing gradient problem[2], an *addition skip connection* similar to that of ResNet [19] is used.

[2] The gradient becomes very small, which prevents the weights from changing and slows down the learning process.

– To further optimize the SqueezeNext architecture, its performance is evaluated using embedded systems (localizing performance bottlenecks, etc.). Based on the obtained results, SqueezeNext is tweaked to achieve higher performance in terms of speed, energy consumption, and classification accuracy.

D) ShuffleNet [43]: this last architecture, which is very efficient in terms of calculation, was specifically designed for mobile devices that have very limited computing powers. To considerably reduce the calculation costs while maintaining accuracy, ShuffleNet architecture is based on two main operations: (1) pointwise group convolution and (2) channel shuffle. In grouped convolution, the filters are partitioned into different groups, each one responsible for a conventional convolution with a certain depth. Channel shuffling mixes up the information from different filter groups by passing from one layer to the next. Overall shuffled grouped convolutions combine the principles of grouped convolutions with those of depth-wise separable convolutions.

2.2 Lightweight communication

Nowadays, being able to equip limited devices with deep learning abilities has become a necessity in many real-world applications, such as smart cities, robotics, autonomous cars, and smart cameras. Besides their features of local processing/deployment on embedded limited systems, optimized CNN architectures offer numerous other advantages like for instance: (1) autonomy and low latency (no need for continuous communications to make a decision). (2) Confidentiality (data can be private and transmitting it to an outside entity may not be possible). (3) Fast exportation of new lightweight models towards customers (e.g., for self-driving cars, new models can be efficiently transmitted from servers to customers' cars. With large models like AlexNet [23], this would require 240MB ore more). (4) Efficient learning (fewer parameters imply faster learning and faster responses).

The IoT technology [16], which is nowadays omnipresent, can connect via the Internet any object that has computing and communication capacities (e.g., sensor devices, cell phones, IP cameras, telematics, etc.). This way, these connected smart devices or objects are given the ability to collaborate to create interesting applications in numerous fields (domotics, health care, automotive, etc.). To concretize the second part of the proposed system (i.e., the connectivity/notification part), we have opted for MQTT[3]; one of the most widely adopted IoT communication protocols [28]. This application-layer protocol is designed to be simple, lightweight, and easy to implement. The goal is allowing devices with weak storage and processing capacities to communicate with one another over low-bandwidth and unreliable networks.

More specifically, MQTT is based on a publish/subscribe messaging paradigm that encompasses three communicating entities: publishers, subscribers, and

[3] MQTT: Message Queuing Telemetry Transport.

a broker. First, publishers publish about some specific topics (i.e., send messages) to the MQTT broker. Second, subscribers can express their subscription interest in any proposed topic at the broker level. That is, a client can subscribe to a topic by sending a *subscribe* message to the broker with the desired QoS (explained below). Clients can also unsubscribe from any topic. Finally, as regards the broker, it handles the publish/subscribe connections by forwarding the received messages from publishers to the appropriate interested subscribers according to their subscriptions.

The MQTT protocol allows communicating data between multiple publishers and multiple subscribers via its message broker while offering three different levels of quality of service (0, 1, and 2). Each of these three levels specifies how the publisher-broker and broker-subscriber communications will take place. In the first level (*QoS 0 - at most once*), the transmitter sends its message only once without waiting for any acknowledgment or confirmation from the receiver. The second level (*QoS 1 - at least once*) ensures that messages are delivered at least once by requesting an acknowledgment (confirmation) for every sent message. In other words, messages, in this case, can be sent/received multiple times. The last level (*QoS 2 - exactly once*) uses a four-step handshake process and guarantees that messages are delivered only once to the corresponding recipient(s). It should be mentioned that a high QoS level does not mean more reliability but also means both higher latency and wider bandwidth.

As it will be thoroughly explained in the next section, in our system, the MQTT broker is utilized to connect smart parking lots (publishers) to end-users/subscribers (car screens, smartphones, or any other smart devices).

Before moving to the next section, we point out that security is a very important aspect of the IoT world. Because connecting personal (daily) objects (such as cars, houses, apparatuses, etc.) to the Internet renders them vulnerable to numerous security threats/risks. So, to be trustworthy and beneficial to end-users, the (smart) IoT applications/infrastructures must ensure both privacy and security. Nonetheless, unlike their classical more powerful counterparts, small constrained IoT devices need new solutions in this regard (i.e., security and privacy). To respond to this need, several approaches have been proposed in the literature. In the following, we start by presenting some solutions that have addressed the use of security frameworks to secure communications in the IoT [30][35][37]. Then, we briefly talk about applying cryptographic schemes to MQTT communications [29][31].

For instance, the work in [30] categorizes numerous privacy-enhancing technologies, potential privacy risks, threats, as well as leakages related to different IoT use cases. The authors also assess the suitability of these technologies for privacy-requiring services within the IoT realm. As for [37], actually, in this work, the authors discuss various IoT security threats and challenges, which emerge due to the vulnerability of both embedded resource-limited systems and utilized communication technologies. The authors also provide an overview of emerging IoT security technologies and state-of-the-art IoT security research trends.

As regards the MQTT security, for instance, to contribute in this direction, in [29] the authors have proposed a novel security framework that aims to provide secure privacy-friendly IoT services. The proposed solution provides three security levels: (1) lightweight data exchanges of non-tampered messages, (2) privacy protection of data sources and receivers, and (3) robust long-term security with mutual authentication for all parties. As a second and last example, we mention that the authors in [31] focused on how to enhance the MQTT security while considering different facts such as (1) the basic MQTT has very limited authentication capabilities and (2) does not support encryption unless using secure tunnels like TLS (Transport Layer Security), which is not suitable for low-power devices.

3 System Description

This section presents the overall architecture and operation principle of our system. We recall that our proposal has been divided into two main complementary stages: (1) periodic lightweight deep learning phase responsible for automatic detection of empty spots in each of the considered/connected parking lots, and (2) instantaneous IoT communication phase which takes care of real-time notifications through a lightweight MQTT publish/subscribe protocol (Figure 1). These two phases are presented in the following two subsections.

3.1 Parking lot connectivity

As depicted in Figure 1, the communication paradigm of the proposed system is composed of three main entities: smart parking lots (publishers), vehicles (drivers or subscribers), and an MQTT broker. The task of drivers notification, or more exactly, the interaction between parking lots, broker, and drivers' applications is done as follows. Each smart connected parking lot periodically determines its number of parked vehicles and checks whether there has been a change (increase or decrease) in the number of empty spots (this operation will be detailed in the next subsection 3.2). If a modification has taken place since the last sampling, then only, in this case, the MQTT broker will be informed (the publisher will report its new number of empty spots). In other terms, the publishing services of parking lots remain idle and communicate with the broker only in the case of changes.

The mission of the broker consists of distributing data received from parking lots to drivers. More specifically, the broker forwards automatically any received information to the interested subscribers (drivers) that have already been successfully authenticated by the system. This way, drivers will receive an up-to-date number of empty spots in any given parking lot of their choice.

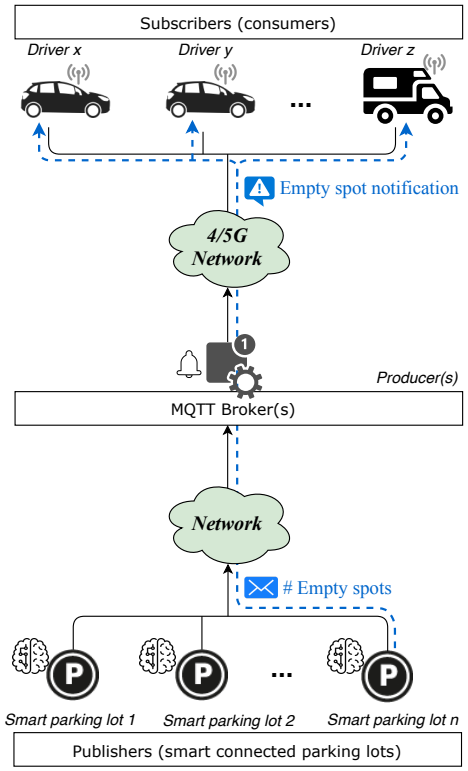

Fig. 1 System overall architecture.

3.2 Parking lot smartness

The proposed parking lot ecosystem has three major components (Figure 2): (1) image acquisition (eyes), (2) deep learning processing (mind), and (3) publishing service (tongue or communication tool). The goal is to efficiently process images that are periodically captured by the various installed cameras (that cover the different parking lots spread over any city) and notify the broker(s) about any change in the number of empty spaces. No extra apparatuses must be needed. That is, this whole process must be ensured by the smart cameras themselves or by the help of their associated low-cost embedded systems such as Raspberry-Pi's. Accordingly, as previously stated, the considered deep-learning technique must be a recent efficient architecture that was specially designed or optimized for mobile and resource-constrained devices.

More in detail, the first component of the smart parking system (i.e., image acquisition) feeds, in a periodic fashion, the deep learning service with images of the monitored parking lot (Figure 2). The deep learning service receives these images as input and produces the estimated number of empty spots in the parking lot. As shown in Figure 3, the adopted deep learning structure is in turn divided into three main parts: (a) model initialization (dataset creation,

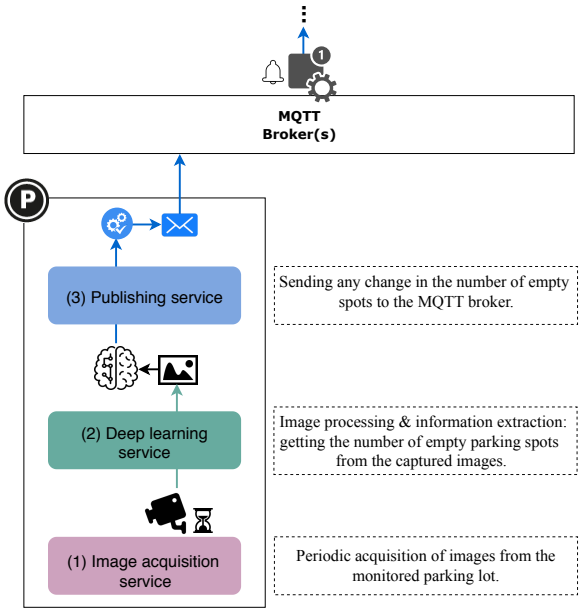

Fig. 2 Smart parking lot architecture.

annotation, etc.), (b) model training, and (c) model utilization (prediction phase). The first phase consists of preparing the chosen deep learning model by collecting and annotating the necessary parking-lot images. As for the second *training phase*, as its name implies, its role is to take the provided dataset of annotated images and train the deep neural network (extracting image characteristics, adjusting weights, etc.). Once the model has been properly trained, it will be continuously utilized to consume the periodically provided parking lot images and produce the desired prediction (decide the existence/absence of vehicles and determine the number of empty spots).

4 System Implementation and Evaluation

This section describes the implemented small-scale version of our proposal and also presents the conducted performance tests and obtained results. To ease its reading, we start, in subsection 4.1, by presenting the configuration and settings of our experimental environment (i.e., the chosen MQTT broker, parking lot setup, and drivers' application). Then, in subsection 4.2, we will provide the obtained evaluation results along with their corresponding interpretation and analysis.

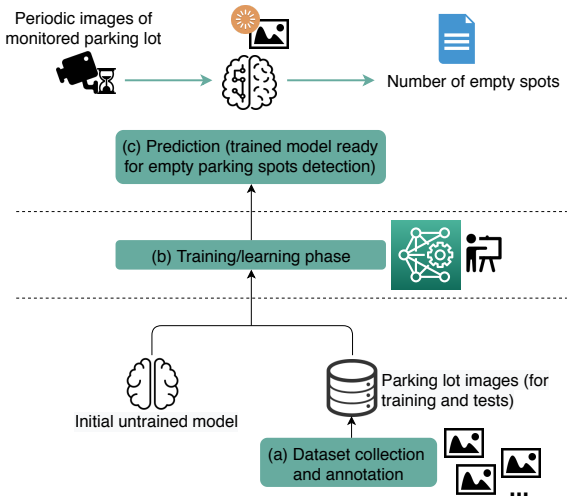

Fig. 3 Three main components of the deep learning service (initialization, training, and utilization).

4.1 System implementation

To assess the operation of the proposed smart IoT system, we implemented an experimental version in which we considered one parking lot, numerous clients/drivers, and one MQTT message broker. The three following subsections will give the details of each of these three main entities.

4.1.1 MQTT message broker

We have opted for the well-known Eclipse Mosquitto broker [32], which has numerous interesting features. This broker, written in C, is designed to (i) have a lightweight nature of carrying out messaging using the publish/subscribe paradigm[4], and (ii) to be suitable for resource-constrained devices (embedded microcontrollers, low-power sensors, smartphones, etc.). Mosquitto is free and proposed for Raspberry Pi, Windows, Mac, and Linux distributions (Debian and Ubuntu). Finally, Mosquitto is open source and provides a library that facilitates the operation of implementing, launching, and connecting *several* MQTT publishers and subscribers. Developers are thus given the ability to adapt and modify the behavior of their systems according to their preferences and needs.

As previously explained, the role of the broker in our system consists simply of (1) waiting for the reception of new numbers of empty spots from the publishing services of parking lots, and (2) forwarding this information to the subscribed clients.

[4] Mosquitto implements the MQTT protocol versions 5.0, 3.1.1, and 3.1.

4.1.2 Parking lot configuration

According to the previously established blueprint/architecture (Figure 2), the implemented parking lot system has three main parts: image acquisition, spot occupancy counting, and result publication. These three services were provided by two main hardware components: an IP camera and a Raspberry Pi. Whereas the IP camera was responsible for image acquisition, the Raspberry Pi took care of both the deep learning processing and hosting the Mosquitto publisher. Regarding the hardware details, the used outdoor IP camera had a 60° angle of view[5], and the *Raspberry Pi 3 Model B*[6] contained an ARM-compatible processor of 1.2 GHz, 1 GB RAM, and a (VideoCore IV) GPU graphics card. In the remainder of this subsection, we will detail the three main substeps of the deep learning service, i.e., model initialization, training, and utilization (Figure 3).

- *Model initialization*: we considered three different deep learning models, namely; ResNet-50 [19], YOLOv3 [34], and MobileNetV2 [36].
  - ResNet-50 [19] is a pre-trained CNN that, as its name indicates, has a 50 layer deep network and can identify 1000 different objects (animals, pencils, cars, etc.).
  - YOLOv3 [34] is the third algorithm of the YOLO[7] family that is able of detecting small objects and aims to further enhance the detection accuracy.
  - MobileNetV2 [36] is a new architecture that is proven to enhance the performance of state-of-the-art mobile models in multiple aspects, especially the efficiency and suitability for resource-constrained devices. MobileNet architectures are similar to regular convolutional neural networks, except for the convolution part, in which MobileNets utilize separable convolutions (*depthwise* and *pointwise* ones). More details about these architectures have been provided in Section 2.1.2.
    The necessary dataset creation for the MobileNet model has been divided into *collection* and *annotation* of parking lot images. For the first stage (i.e., data collection), we utilized images from PKLot and CNRPark-EXT datasets [10,11], and also considered numerous other parking lot images that were taken by our research team (Figure 4). For the second stage (i.e., images annotation), we chose LabelImg; a graphical annotation tool[8].
- *Model training*: the considered non-trained MobileNet model was trained with 2500 iterations for a total processing time of 60 hours. The utilized computer (PC) for this task had an Intel i5 CPU, and 8GB of RAM.
- *Model utilization*: once ready, each of the three considered deep learning models has been utilized to receive parking lot images as input and pro-

[5] `http://www.aquilavizion.com/smartvizion_ip_mini_av-ipe08hd/`
[6] `https://www.raspberrypi.org/`
[7] You Only Look Once.
[8] `https://github.com/tzutalin/labelImg/`

Fig. 4 Example of training images.

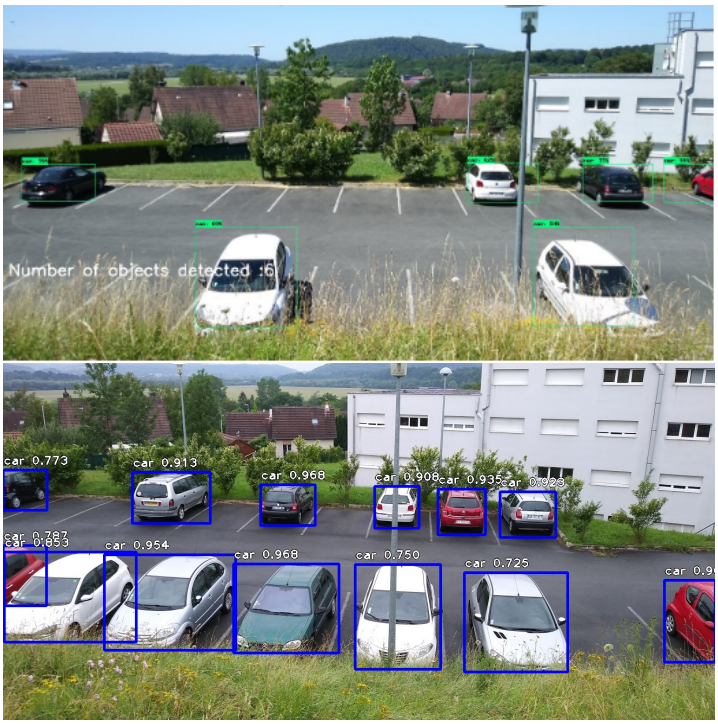

Fig. 5 Example of processing results obtained by the Raspberry Pi using MobileNetV2 (top) and ResNet-50 (bottom) models.

duce the number of parked vehicles. Figure 5 provides an example in which the considered MobileNetV2 and ResNet-50 architectures have successfully detected the existing cars. The obtained numbers of empty spots are then exploited by the publishing MQTT service, which was programmed to pub-

lish information about one topic ("*empty-spots*") to which drivers will be automatically/implicitly subscribed to upon authentication.

4.1.3 Drivers' application

The last tier of our system was realized as an Android application that allows end-users to navigate through a map and display the connected parking lots along with their corresponding up-to-date number of empty spots. As shown in Figure 6, once authenticated, drivers can also choose to receive/display real-time images of any chosen parking lot. Indeed, when drivers register and log in to the system (via their smartphones, car screens, etc.), they implicitly express their interest in the already defined topic "empty-spots" which is available at the broker level. Finally, in order not to burden the broker, when a driver exits the application, it will be implicitly unsubscribed and disconnected from the broker.

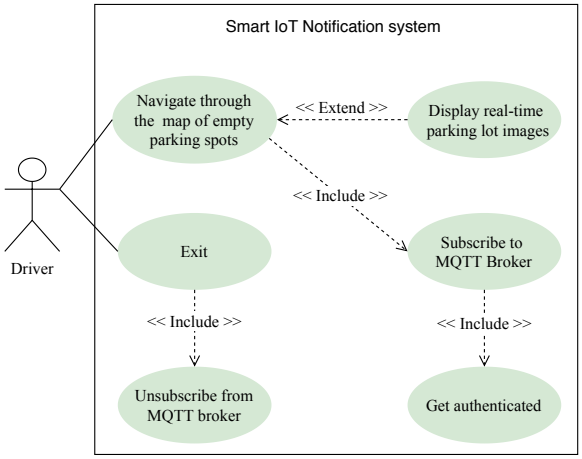

Fig. 6 Behavior of the developed Android application.

4.2 System evaluation

The adopted first-order testing started as follows. After launching the Mosquitto broker, the parking lot publishing service was also created and connected to the latter. After a while, the smart parking lot system, described in Section 3.2, started its periodic estimation of empty spots. Finally, vehicles (or more exactly, copies of the developed mobile application) were gradually launched, connected to the broker, and tested.

In addition to testing the smooth operation and effectiveness of our system in terms of real-time notifications, we have also evaluated and compared the

lightweight deep learning architecture that we have trained (MobileNetV2 [36]) against the conventional ones (ResNet-50 [19] and YOLOv3 [34]) and compared other pre-trained lightweight architectures against each other (Tiny-YOLO [34], SSD-MobileNetV2 [36], and SqueezeDet[40]). We start in subsection 4.2.1 by presenting the results that were obtained while comparing MobileNetV2 [36], ResNet-50 [19], and YOLOv3 [34] using (1) a Raspberry Pi 3 Model B and (2) a computer (PC) that has an Intel i5 CPU, 8GB RAM, and an NVIDIA GeForce GT 720M graphics card. Then, in subsection 4.2.2, we will present the results that were obtained by comparing Tiny-YOLO [34], SSD-MobileNetV2 [36], and SqueezeDet[40] with each other and with a pruned pre-trained SqueezeDet architecture.

4.2.1 Lightweight versus conventional deep learning

According to Table 1 (PC results), MobileNetV2 [36] is the most effective in terms of processing time with less than one second (or more exactly 0.5 seconds on average). ResNet-50 [19] comes in second place with 7 seconds, while YOLOv3 [34] arrives last with 13 seconds. As regards the number of detected vehicles, all three models have unstable estimation performance that is sometimes good and other times bad. Figure 7 shows a case in which MobileNetV2 fails to detect all the existing cars.

Table 1 Prediction time and precision: PC results (num-DV represents the number of detected vehicles).

ResNet-50		**YOLOv3**		**MobileNetV2**	
num-DV	**Time (s)**	**num-DV**	**Time (s)**	**num-DV**	**Time (s)**
0	9.26	2	13.08	100	2.27
4	7.86	0	13.05	15	0.27
1	7.85	0	13.12	7	0.20
1	7.85	9	13.08	75	0.33
3	7.88	3	13.03	75	0.35
2	7.83	2	13.04	82	0.36
2	7.85	3	13.02	80	0.31
7	7.06	7	13.14	7	0.95
6	7.06	6	13.21	7	0.79
13	7.06	13	13.00	11	0.26
4	7.04	4	13.07	6	0.21
11	7.03	12	13.24	12	0.24
8	7.03	8	13.21	6	0.76
9	7.02	9	13.15	10	0.73
5	7.86	9	13.08	4	0.23
12	7.89	12	13.02	3	0.24
1	7.86	1	13.05	2	0.18
30	7.85	34	13.10	13	0.25
46	7.94	46	13.11	11	0.23
31	7.89	36	13.05	15	0.27

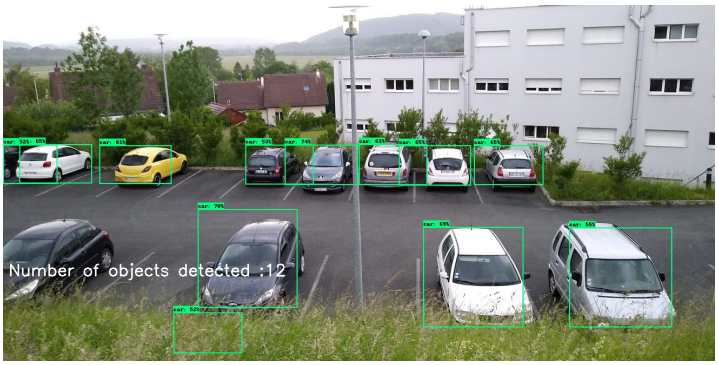

Fig. 7 Example of imprecise MobileNetV2 results.

According to Table 2 (Raspberry Pi results), MobileNetV2 is also more effective in terms of processing time (with two seconds on average). Indeed, when compared with the PC results of Table 1, the performance of MobileNetV2 has not been affected by the limited resources of the Raspberry Pi. This confirms the claims that MobileNet is suitable for mobile resource-constrained systems. Concerning ResNet-50, in this scenario, it also comes second but with more than 100 seconds (against 7 seconds with the PC). These results show that ResNet-50 is considerably affected and requires more computational power to be faster. In other words, due to its capacities, the PC took less time (than the Raspberry Pi) to respond to the requirements of ResNet-50. However, this was not the case for MobileNet which stands out with its lightness. Finally, as for YOLOv3, it is absent in Table 2 because, on one hand, the Raspberry Pi is very limited in terms of storage and computing, and on the other hand, YOLOv3 is very demanding in terms of resources, especially when compared with ResNet-50 and MobileNetV2.

Based on the obtained results, we conclude that MobileNet is more efficient in terms of time and resource consumption. For this reason, we suggest opting for lightweight deep learning architectures. This way, the utilized small-sized resource-constrained embedded systems will not be overwhelmed by the deep learning service.

4.2.2 Lightweight deep learning evaluation

We recall that the three pre-trained lightweight architectures that we have considered are Tiny-YOLO [34], SSD-MobileNetV2 [36], and SqueezeDet[40]. As demonstrated in the previous subsection 4.2.1, the conventional YOLOv3 is not optimal nor fast to be run on embedded devices such as the Raspberry Pi. To remedy this deficiency, the developers of YOLO (Redmon et al. [34]), proposed a variation of this architecture called Tiny-YOLO. The small size and fast inference of this solution make it suitable for embedded computer vision devices. As for SqueezeDet [40], it adopts a single pass detection pipeline (the proposition and classification of regions are carried out simultaneously by a

Table 2 Prediction time and precision: Raspberry Pi results (num-DV is the number of detected vehicles).

ResNet-50		**MobileNetV2**	
num-DV	**Time (s)**	**num-DV**	**Time (s)**
0	101.12	100	7.06
3	121.15	15	1.28
1	114.72	7	0.99
1	117.91	75	1.43
3	118.94	75	1.47
2	120.04	82	1.47
2	120.77	80	1.46
7	107.76	7	3.73
6	108.57	7	3.73
13	107.84	11	1.13
4	108.96	6	0.94
11	108.95	12	1.16
8	109.14	6	3.36
9	108.60	10	4.61
5	121.73	4	0.85
12	122.90	3	0.82
1	123.23	2	0.78
31	123.62	13	1.21
46	123.54	11	1.14
31	123.79	15	1.28

single network). SqueezeDet can consider several CNN architectures to extract the characteristics (ResNet, etc.). However, as the objective is to minimize the size, accelerate inferences, and minimize energy consumption, the developers of SqueezeDet [40] have opted for SqueezeNet [22] as a basic architecture (details about SqueezeNet are provided in Section 2.1.2).

The obtained results, depicted in Figure 8, show the outperformance of SqueezeDet in terms of inference time, and this in comparison to both MobileNetV2 and Tiny-YOLO. Being able to predict/process images with high resolutions is one of the most important factors in improving the performance of this model (the utilized evaluation images have a 1248×384 resolution). To make the original SqueezeDet model more efficient, we have optimized it using a pruning technique called ℓ_1-*norm* [26] (Section 2.1.1). In brief, this method determines and prunes the least useful filters of an already trained model. Figure 8 shows that the pruned SqueezeDet is faster than the original architecture.

Figure 9 depicts the effect of ℓ_1-*norm* pruning on both the *size* and *precision* of the basic SqueezeDet. In more exact words, we measured the pruning effect on the model size as well as on the four following performance metrics: loss, precision, recall, and average precision, which are the most used metrics to evaluate object detector mechanisms. The performed tests were run using the entire Kitti database [14] with 80% of the images for fine-tuning and 20% for the evaluation. In brief, Kitti [14] is an object-detection dataset that includes images and bounding rectangles. In more exact words, this set

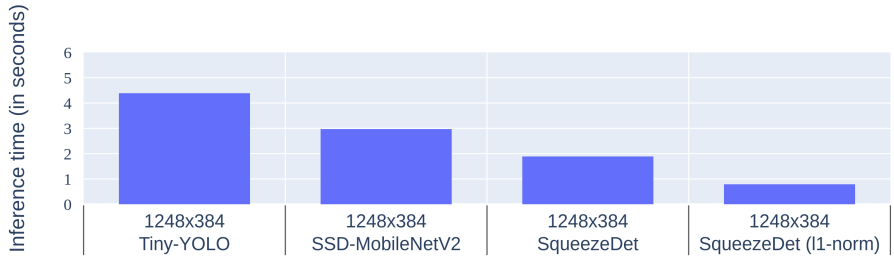

Fig. 8 Comparison in terms of inference time: Tiny-YOLO vs. SSD-MobileNetV2 vs. SqueezeDet.

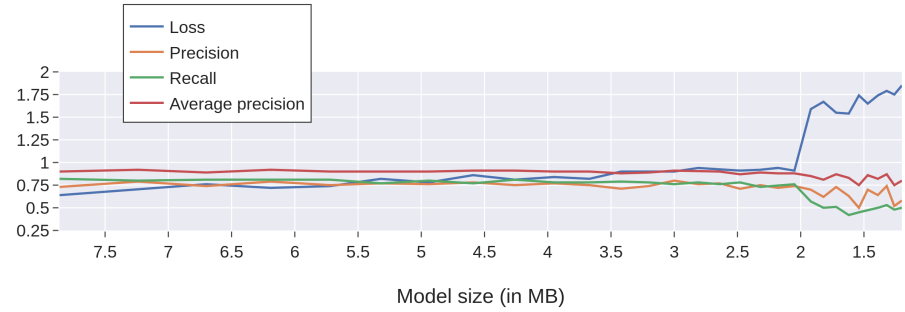

Fig. 9 ℓ_1-*norm* pruning effect on both the size and precision of SqueezeDet.

contains 7481 training-images and 7518 test-images annotated with bounding rectangles containing 80256 objects belonging to 8 different classes (cars, trucks, trams, vans, pedestrians, persons, cyclists, and "dont care"). A full description can be found on Kittis homepage[9].

The obtained results (Figure 9) show that the performed pruning does not affect the precision of the basic model; the loss remains tolerable until the model size becomes approximately 2MB. Beyond this point, the performance degradation starts to become apparent. The initial size of SqueezeDet is equal to 8.5MB, its precision $P_1 = 0.84$, recall $R_1 = 0.84$, loss $L_1 = 0.4$, and average precision $AP_1 = 0.93$. The pruned optimal model has a size of 2.05MB, an accuracy $P_2 = 0.74$, recall $R_2 = 0.76$, loss $L_2 = 0.91$ and an average precision $AP_2 = 0.88$. Accordingly, when compared with the initial model, we lost 0.1 in Precision, 0.08 in Recall, 0.51 in Loss, and 0.05 in AP. Compared to this insignificant difference, the applied optimization has led to a 4.14× reduction in the model size.

Finally, Figure 10 shows the effect of the pruned SqueezeDet (the effect of its size) on the time required to make a prediction using a Raspberry Pi 3

[9] cvlibs.net/datasets/kitti/

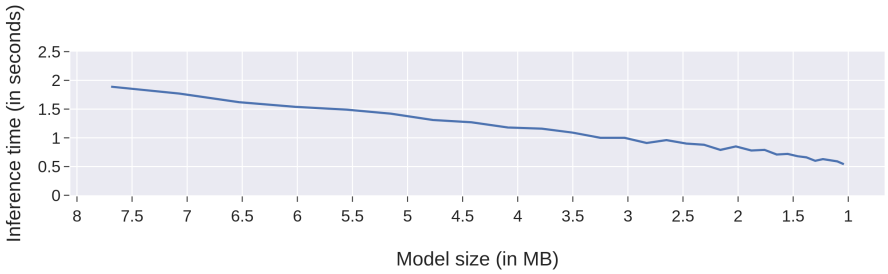

Fig. 10 ℓ_1-*norm* pruning effect on the inference time of SqueezeDet.

Model B. The depicted inference time is the average of 20 predictions per iteration (each prediction is performed on a single thread and a different image). The inference time of the original Squeeze model is equal to 1.89 seconds and that of the pruned optimal one is equal to 0.79 seconds. That is to say, the latter is 2.39 times faster than the original one. Also, as previously shown in Figure 8, the pruned SqueezeDet is 3.75 times faster than SSD-MobileNetV2 and 6.21 times faster than Tiny-YOLO.

5 Related Work

In the following, we will succinctly review some parking assistance solutions related to both parking-spots availability and parking-lots monitoring. First, the technique proposed in [12] by Delibaltov et al. models each parking spot as a volume in a 3D space, which allows it to take into account occlusions when estimating the presence of a vehicle in a parking space. The solution of Amato et al. [3], proposed for automatic detection of empty spots in parking lots, creates a neural network mAlexnet. The idea is to be able to detect the availability of each place in the parking using a mask (place coordinates). Ahrnbom et al. [1] proposed a fast lightweight feature extraction technique that can be run on resources limited devices. They formulated the problem as a binary classification: parked car and empty slot, using the logistic regression algorithm. They further used SVM (Support Vector Machine) for classification purposes. The authors in [42] designed and implemented a smart parking system based on wireless sensor networks. In this system, each parking spot is equipped with a sensor node that detects its availability and sends this information to an embedded Web-server which forwards it to a central Web-server using Wi-Fi networks in a real-time manner. The driver can then track the status of a parking spot thanks to a mobile phone application. The authors in [44] exploit UAVs (Unmanned Aerial Vehicles) to find the number of cars in a parking area. They approached the car counting problem as a density estimation of pixel values. They also used the SIFT algorithm (Scale-Invariant Feature Transform) to extract car features, which are then grouped

using K-Mean clustering to generate features codebook. Finally, they trained their system using pre-collected UAV images.

To deal with the issue of finding a parking space, instead of building new parking lots (which costs a lot, destroys nature, and sometimes is not possible due to the lack of space in cities), the authors in [38] propose to optimize old ones by exploiting the evolution of IoT and AI. More specifically, Shoeibi et al. proposed an eco-friendly system called Automated Valet Parking, which helps optimizing parking space usage with Deep Q-Learning (a reinforcement learning method).

To address/predict the vehicle-park occupancy rate in smart cities, the authors in [9] proposed a new technique based on Recurrent Neural Networks. In brief, their approach consists of designing a predictor (deep network) that encapsulates the behavior of car occupancy and thus can make an informed guess on the number of free parking spaces.

In [41], Xiang et al. propose a novel vision-based parking occupancy detection method based on a Haar-AdaBoosting algorithm and a convolutional neural network (CNN). The proposed technique aims to achieve real-time detection of vehicles with high accuracy. To do so, first, the Haar-AdaBoosting classifier is utilized to obtain a set of sub-windows, which might contain vehicle regions. Then, these sub-windows are passed to the adopted CNN to filter non-vehicle regions. Finally, to recognize/determine the occupied parking spaces (as well as illegal occupancies), the authors proposed their specific method for parking occupancy detection.

To remedy the issues related to sensor-based parking systems (such as the non-reliability and expensiveness), the work of Bachani et al. [6] presents a comprehensive analysis and focuses on the crucial aspects of designing a smart parking system (such as sensor selection and optimal sensor deployment). For instance, the authors evaluated the use of two different types of sensors, namely; Light Dependable Resistor (LDR) sensors (which work on shadow detection principal) and Infra-Red (IR) sensors (which work on object detection mechanism). To assess its accuracy (detection of vacant parking slots/vehicles), the system was tested under different conditions and environmental factors. The obtained results confirmed that the IR sensors outperform the LDR ones in terms of accuracy.

In addition to approaches that use visual techniques and ground-based sensors, there exist some other solutions which are based on sensors that are installed in cars or carried by drivers. For instance, Caicedo et al. [8] proposed a solution that aims to predict parking occupancy by interacting with the onboard navigation systems. Another example of this category is the solution proposed by Lan et al. [24], which is based on smartphone sensors and mainly aims to collect real-time parking availability information.

Finally, we mention that some works have addressed the car parking problem while considering large cities and implementing new parking systems (intelligent mobile applications, etc.) that aim to organize the randomness in parking management (to reduce the congestion in some areas, reduce the time dedicated/wasted to searching for parking spots, etc.). As examples, we cite

for instance, Abu Dhabi [2] and California [33] (the areas of San Francisco and Los Angeles).

6 Conclusions and Future Work

To address the *in-city vehicle parking* problem, we proposed in this paper a smart IoT system that was designed to efficiently manage parking lots of large densely-populated cities. The proposed system can be easily installed and extended. First, using efficient lightweight deep learning, the system counts empty spots in a whole city or any designated parking lot. Second, using the well-known lightweight MQTT protocol, it instantaneously notifies its users. To assess the performance of this system, a small-scale version of it was implemented and evaluated while considering different lightweight and conventional deep learning solutions (MobileNetV2, ResNet-50, YOLOv3, Tiny-YOLO, SSD-MobileNetV2, SqueezeDet, and SqueezeDet pruned with ℓ_1-$norm$).

As future directions, we are currently working on several techniques that aim to optimize/compress deep neural networks. Indeed, in this paper, we considered a small-scale testbed, so as future research directions, we are also planning to analyze the scalability of the proposed system in the case of multi-parking lots as well as multi-camera scenarios.

Acknowledgements The authors would like to thank Mohammed Habib Allah Kechout and Sebti Tamraoui, from the Higher National School of Computer Science (Algiers), for their precious help to this work.

References

1. Ahrnbom, M., Astrom, K., Nilsson, M.: Fast classification of empty and occupied parking spaces using integral channel features. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 9–15 (2016)
2. Alkheder, S.A., Al Rajab, M.M., Alzoubi, K.: Parking problems in abu dhabi, uae toward an intelligent parking management system adip: Abu dhabi intelligent parking. Alexandria Engineering Journal **55**(3), 2679–2687 (2016)
3. Amato, G., Carrara, F., Falchi, F., Gennaro, C., Meghini, C., Vairo, C.: Deep learning for decentralized parking lot occupancy detection. Expert Systems with Applications **72**, 327–334 (2017)
4. Anwar, S., Hwang, K., Sung, W.: Fixed point optimization of deep convolutional neural networks for object recognition. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1131–1135. IEEE (2015)
5. Anwar, S., Hwang, K., Sung, W.: Structured pruning of deep convolutional neural networks. ACM Journal on Emerging Technologies in Computing Systems (JETC) **13**(3), 1–18 (2017)
6. Bachani, M., Qureshi, U.M., Shaikh, F.K.: Performance analysis of proximity and light sensors for smart parking. Procedia Computer Science **83**, 385–392 (2016)
7. Belbachir, A.N.: Smart cameras, vol. 2. Springer (2010)
8. Caicedo, F., Blazquez, C., Miranda, P.: Prediction of parking space availability in real time. Expert Systems with applications **39**(8), 7281–7290 (2012)
9. Camero, A., Toutouh, J., Stolfi, D.H., Alba, E.: Evolutionary deep learning for car park occupancy prediction in smart cities. In: International Conference on Learning and Intelligent Optimization, pp. 386–401. Springer (2018)

10. Database, P.L.: http://web.inf.ufpr.br/vri/databases/parking-lot-database/ (2020)
11. De Almeida, P.R., Oliveira, L.S., Britto Jr, A.S., Silva Jr, E.J., Koerich, A.L.: Pklot–a robust dataset for parking lot classification. Expert Systems with Applications **42**(11), 4937–4949 (2015)
12. Delibaltov, D., Wu, W., Loce, R.P., Bernal, E.A.: Parking lot occupancy determination from lamp-post camera images. In: 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013), pp. 2387–2392. IEEE (2013)
13. Ge, Z., Bewley, A., McCool, C., Corke, P., Upcroft, B., Sanderson, C.: Fine-grained classification via mixture of deep convolutional neural networks. In: 2016 IEEE Winter Conference on Applications of Computer Vision (WACV), pp. 1–6. IEEE (2016)
14. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the kitti vision benchmark suite. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 3354–3361. IEEE (2012)
15. Gholami, A., Kwon, K., Wu, B., Tai, Z., Yue, X., Jin, P.H., Zhao, S., Keutzer, K.: Squeezenext: Hardware-aware neural network design. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2018, Salt Lake City, UT, USA, June 18-22, 2018, pp. 1638–1647 (2018)
16. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (iot): A vision, architectural elements, and future directions. Future generation computer systems **29**(7), 1645–1660 (2013)
17. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: International Conference on Machine Learning, pp. 1737–1746 (2015)
18. He, K., Sun, J.: Convolutional neural networks at constrained time cost. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 5353–5360 (2015)
19. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
20. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
21. Huang, Q., Zhou, K., You, S., Neumann, U.: Learning to prune filters in convolutional neural networks. In: 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), pp. 709–718. IEEE (2018)
22. Iandola, F., Han, S., Moskewicz, M., Ashraf, K., Dally, W., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡ 0.5 mb model size. february 2016. arXiv preprint arXiv:1602.07360 (2019)
23. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp. 1097–1105 (2012)
24. Lan, K.C., Shih, W.Y.: An intelligent driver location system for smart parking. Expert Systems with Applications **41**(5), 2443–2456 (2014)
25. LeCun, Y., Boser, B.E., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W.E., Jackel, L.D.: Handwritten digit recognition with a back-propagation network. In: Advances in neural information processing systems, pp. 396–404 (1990)
26. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: 5th International Conference on Learning Representations (2017)
27. Lin, D., Talathi, S., Annapureddy, S.: Fixed point quantization of deep convolutional networks. In: International conference on machine learning, pp. 2849–2858 (2016)
28. Locke, D.: Mq telemetry transport (mqtt) v3. 1 protocol specification. IBM developerWorks Technical Library **15** (2010)
29. Malina, L., Srivastava, G., Dzurenda, P., Hajny, J., Fujdiak, R.: A secure publish/subscribe protocol for internet of things. In: Proceedings of the 14th International Conference on Availability, Reliability and Security, pp. 1–10 (2019)
30. Malina, L., Srivastava, G., Dzurenda, P., Hajny, J., Ricci, S.: A privacy-enhancing framework for internet of things services. In: International Conference on Network and System Security, pp. 77–97. Springer (2019)
31. Mathews, S.P., Gondkar, R.R.: Protocol recommendation for message encryption in mqtt. In: 2019 International Conference on Data Science and Communication (IconDSC), pp. 1–5. IEEE (2019)

32. an open source MQTT broker, E.M.: https://mosquitto.org/ (2020)
33. Rajabioun, T., Ioannou, P.A.: On-street and off-street parking availability prediction using multivariate spatiotemporal models. IEEE Transactions on Intelligent Transportation Systems **16**(5), 2913–2924 (2015)
34. Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767 (2018)
35. Roukounaki, A., Efremidis, S., Soldatos, J., Neises, J., Walloschke, T., Kefalakis, N.: Scalable and configurable end-to-end collection and analysis of iot security data: Towards end-to-end security in iot systems. In: 2019 Global IoT Summit (GIoTS), pp. 1–6. IEEE (2019)
36. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4510–4520 (2018)
37. Sezer, S.: T1c: Iot security:-threats, security challenges and iot security research and technology trends. In: 2018 31st IEEE International System-on-Chip Conference (SOCC), pp. 1–2. IEEE (2018)
38. Shoeibi, N., Shoeibi, N.: Future of smart parking: Automated valet parking using deep q-learning. In: International Symposium on Distributed Computing and Artificial Intelligence, pp. 177–182. Springer (2019)
39. Shoup, D.C.: Cruising for parking. Transport Policy **13**(6), 479–486 (2006)
40. Wu, B., Iandola, F., Jin, P.H., Keutzer, K.: Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 129–137 (2017)
41. Xiang, X., Lv, N., Zhai, M., El Saddik, A.: Real-time parking occupancy detection for gas stations based on haar-adaboosting and cnn. IEEE Sensors Journal **17**(19), 6360–6367 (2017)
42. Yang, J., Portilla, J., Riesgo, T.: Smart parking service based on wireless sensor networks. In: IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society, pp. 6029–6034. IEEE (2012)
43. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, pp. 6848–6856 (2018)
44. Zhou, H., Wei, L., Fielding, M., Creighton, D., Deshpande, S., Nahavandi, S.: Car park occupancy analysis using uav images. In: 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 3261–3265. IEEE (2017)