sensors

MDPI

Review

Edge Machine Learning for AI-Enabled IoT Devices: A Review

Massimo Merenda [1,2,*], Carlo Porcaro [1,2] and Demetrio Iero [1,2]

[1] Department of Information Engineering, Infrastructure and Sustainable Energy (DIIES), University Mediterranea of Reggio Calabria, 89124 Reggio Calabria, Italy; porcarocarlo@libero.it (C.P.); demetrio.iero@unirc.it (D.I.)

[2] HWA srl-Spin Off dell'Università Mediterranea di Reggio Calabria, Via Reggio Campi II tr. 135, 89126 Reggio Calabria, Italy

* Correspondence: massimo.merenda@unirc.it; Tel.: +39-0965-1693-441

Received: 22 March 2020; Accepted: 25 April 2020; Published: 29 April 2020

Abstract: In a few years, the world will be populated by billions of connected devices that will be placed in our homes, cities, vehicles, and industries. Devices with limited resources will interact with the surrounding environment and users. Many of these devices will be based on machine learning models to decode meaning and behavior behind sensors' data, to implement accurate predictions and make decisions. The bottleneck will be the high level of connected things that could congest the network. Hence, the need to incorporate intelligence on end devices using machine learning algorithms. Deploying machine learning on such edge devices improves the network congestion by allowing computations to be performed close to the data sources. The aim of this work is to provide a review of the main techniques that guarantee the execution of machine learning models on hardware with low performances in the Internet of Things paradigm, paving the way to the Internet of Conscious Things. In this work, a detailed review on models, architecture, and requirements on solutions that implement edge machine learning on Internet of Things devices is presented, with the main goal to define the state of the art and envisioning development requirements. Furthermore, an example of edge machine learning implementation on a microcontroller will be provided, commonly regarded as the machine learning "Hello World".

Keywords: artificial intelligence; machine learning; Internet of Things; edge devices; deep learning

1. Introduction

The Internet of Things (IoT) scenario [1,2] has gained a lot of notoriety in recent years. It encompasses an infrastructure of software and hardware that connects the physical world with the Internet. Due to the explosive growth of interest in this paradigm, the number of IoT devices has increased dramatically in recent years. It has been estimated that by 2025, more than 75 billion devices will be connected to the Internet [3], leading to an economic impact on the global market. IoT devices typically have limited computing power, small memories, and could generate large amounts of data. Low-power and connected systems including mainly sensors will be used in our homes, cities, vehicles, and industries. Cloud computing might be suitable for the IoT sector growth but the delay caused by the data transfer is unacceptable for some tasks (e.g., health monitoring), in addition to possible bandwidth saturation. So, due to the greater numbers of connected devices, the only-cloud processing could become impractical and would lead to greater latency, bandwidth decrease, and privacy and reliability problems [4]. Hence the need to bring the calculation as locally as possible, incorporating intelligence on end devices to limit cloud traffic. This means giving a sort of "consciousness" to the devices that become able to interact also in the absence of the connection,

elaborating complex behavior, and adapting to rapidly changing situations, a sort of "Internet of Conscious Things". Unfortunately, limitations in the computational capabilities of resource-scarce devices restrict the implementation of complex machine learning (ML) algorithms on them, although several frameworks based on software agents provide reliable and effective solutions for the optimizations of different edge computing implementation [5–7]. The tasks that can be delivered to the edge elements are related to low-data fusion [8], while for a deeper understanding of the data (e.g., decision-making purposes) it is necessary to assign the calculation to more efficient systems. However, the transferring of raw data to cloud servers increases communication costs, causes delayed system response, and exposes private data. To address these issues, a practical solution is to consider processing data closer to its sources and transmitting to remote servers only the data needed for further cloud processing. Edge computing refers to computations being performed as close to data sources as possible, instead of remote locations.

Both search engines and libraries were used to write the review. As reported on Google Trends [9], in these years there has been an increase of interest from the scientific community on the topic and issues of *edge computing* (Figure 1) and a total of 6342 papers are reported from Scopus [10]. The keywords employed to obtain the papers to be analyzed was *edge computing*.

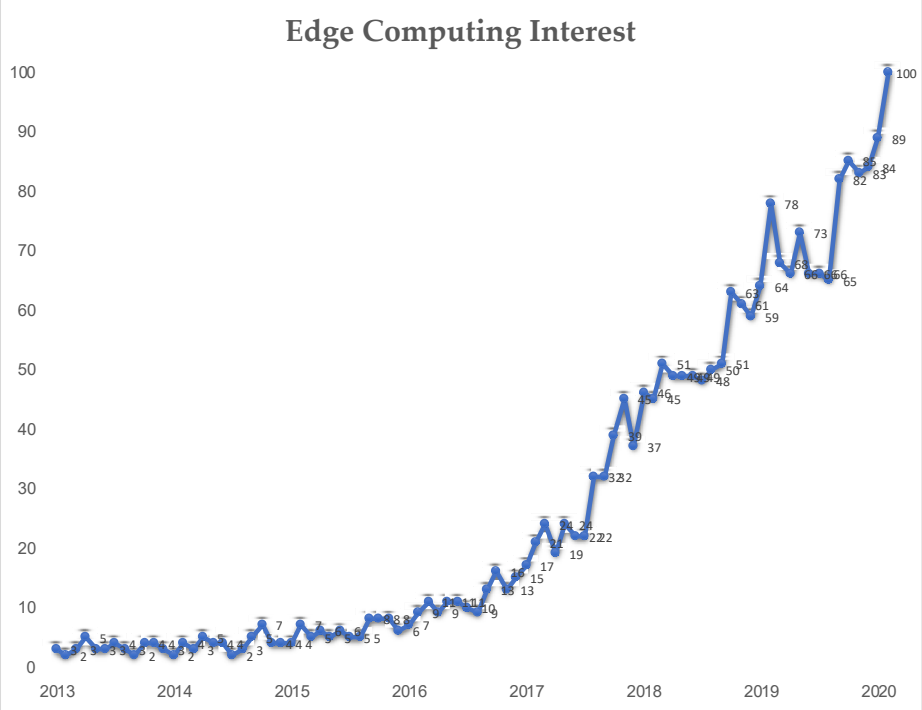

Figure 1. Edge computing interest (Google Trends).

In order to discern which papers to use, only papers written in English were considered. Furthermore, to observe the recent evolution in this field, the selected papers were published from the years 2014 to the first months of 2020. Finally, a total of about 100 papers were utilized to compile this review.

The distribution of the papers regarding the country of the first author providing the countries that investigate edge computing systems are mainly China, USA, UK, and Italy. China is the country with the highest number of papers with a total of 2193 papers, 34.7% of the total.

The solution paved from the usage of edge machine learning (eML) is a viable way to meet the latency, scalability, and privacy challenges described earlier.

In order to make possible this scenario, efficient artificial intelligence (AI) algorithm could be deployed on the devices as SVM (support vector machine) [11,12], deep learning (DL) [13,14] and, *inter alia*, neural networks (NNs) [15,16]. It is noteworthy that NNs require less computational power in the application phase than in the training phase. This feature can be exploited to execute algorithms

of AI on devices with limited resources, as microcontrollers (MCUs), allowing local data processing. In fact, in the learning phase, a large amount of data is used to calculate the weights and biases of the network, thus requiring a performing machine in this phase. Once the learning phase has been completed and the network has been trained, the model can be used for inference (in statistic field, it is the process of using data analysis to deduce properties of an underlying probability distribution) with a lower computational request. In fact, once the coefficients have been calculated, they are stored in the program memory and the AI algorithm can be executed on a device with a low capacity in terms of RAM (random access memory). ML algorithms are used for different topics such as smart cities [5,17–22], computer vision [23–27], health care [28–33], automotive [34–37], and others. Concerning these fields, there are different examples of how machine learning could be brought on edge devices. Anandhalli and Baligar in 2017 proposed in [18] a video processing algorithm that identified and counted the vehicles on a road. The algorithm was run on a Raspberry Pi3 (1.2 GHz quad-core ARMv8, 1 GB of RAM) with a built-in camera, using the library OpenCV5 (Open Source Computer Vision Library [38]). In [23], the authors developed a face recognition algorithm for law enforcement agencies within a smart city. A portable wireless camera mounted on the uniform of a police officer is used to capture the images that are then passed on to a Raspberry Pi3 to perform facial recognition. The Viola-Jones algorithm [39] is first used to identify faces in the images, then the ORB algorithm [40] extracts the peculiar features from the faces that are then transmitted to an SVM algorithm in the cloud to identify people. Therefore, for IoT purposes, the devices must be sufficiently powerful to perform certain tasks, even though, in general, it is possible to insert AI even in any embedded devices by exploiting a certain class of algorithms. The use of ML algorithms also allows the extension of the average battery life of the device, with power saving one of the fundamental tasks of the IoT world. For example, authors in [28] focused on increasing the battery life of a device used for e-health purposes by optimizing sampling times and data transfers using ML algorithms. Any unnecessary data that are transferred, stored, and processed appear to be a potential waste of energy. Using an SVM algorithm, based on the RBF (radial basis function) [41], kernel function, and varying the sampling frequency, it was possible to increase the life of the device from 2 weeks to years (997 days). In the same scenario considered, the wearable sensor platform for health-care in a residential environment (SPHERE) [42] is used to classify human activity into three categories: Sedentary, moderate, and sportive. The data measured by an accelerometer are sent to the MCU using Serial Peripheral Interface (SPI) protocol. After processing the data, they are packed by the microcontroller and sent using *advertisement mode* to a Bluetooth Low Energy (BLE) radio, which transmits data outside the smart home to a central unit. The use of SVM can also be found in [43], which highlights how the use of ML algorithms allows greater efficiency and low consumption in predicting patients' seizures. In fact, crisis prediction is a difficult task due to the variability of the electroencephalography (EEG) signal depending on the patient. A neurostimulator able to identify and react to a principle of crisis can facilitate applications that would not otherwise be feasible, such as the possibility of generating a stimulus to suppress the crisis itself. The article demonstrates how ML techniques like SVM can be used to identify possible crises in each patient. The use of NN on embedded devices can be found in [44], which deals with the use of learning algorithms on an inexpensive robot built to perform sense-motor tasks. The robot, in particular, learns to trace objects by identifying the peculiarities of the object itself. The algorithm uses a CNN (convolutional neural network) to combine color, brightness, motion, and audio information while training is carried out using supervised ML algorithms. The images are properly analyzed to eliminate redundant input data. A Motorola board populated with a 68HC11, an 8-bit MCU with Complex Instruction Set Computer (CISC) architecture is used, which converts and processes the gyroscope outputs and generates sound feedback and timing signals for 14 servomotors that allow 3 degrees of freedom for each of the four legs and two degrees of freedom for the robot's head. A Charge-Coupled Device (CCD) camera is mounted on the top of the robot, which acts as the robot's eye.

As seen from the previous papers, ML algorithms can be implemented on devices with limited computational power and this can be used to improve the IoT field [45,46], thus enabling the *edge computing*. A related term, *fog computing*, describes an architecture where the cloud is extended to be

closer to the IoT end devices, thereby improving latency and security by performing computations near the network edge [4]. Therefore, also for the fog computing, the task is to bring the processing phase closer to where the data are generated but the main difference is where the "intelligence" is located. In fog computing the processing phase is at the LAN (local-area network) level, in a fog node or IoT gateway. In edge computing, data are mainly processed directly on the devices to which the sensors are attached (physically very close to the sensors). The closer to the sensor, the better it is in terms of privacy and power consumption because of the reduction of the energy request related to data transmission. In this scenario, a variety of possibilities of energy harvesting from different sources paves the way for AI-enabled passive or semipassive IoT sensor platforms [47–51].

This survey focused on ML systems deployed on edge devices. Section 2 provides a comparison between the ML algorithms implementable in edge computing. In Section 3, the process of bringing ML to the edge is analyzed. Section 4 describes edge server-based architectures while in Section 5 the wireless standards for AI-enabled IoT devices are introduced. Section 6 provides edge-specific solutions for offloading techniques, detailing the differences of the joint computation alternatives. Section 7 deals with privacy issues and how to protect user privacy in uploading data. Section 8 describes the edge implementations of the training phase, in ML design. In Section 9, an example of edge machine learning implementation is provided, commonly regarded as the machine learning "Hello World". The conclusions are, finally, drawn in Section 10.

2. Machine Learning Algorithms

We now discuss ML algorithms that could be used in resource-constrained settings at the edge of the network. The machine learning algorithms introduced in the next paragraphs are the most used in the papers that afford the problem of bringing AI in devices with resource-constrained hardware.

2.1. Deep Learning

A deep learning model can be thought as a combination of weights and biases [52]. These parameters are varied by an optimization function (ADAM [53] optimization algorithm is generally used) based on an objective function (loss function or reward function, if the learning is, respectively, supervised or reinforcement) that measures the predictive power of the model. Following a training phase, the AI algorithm identifies an underlying pattern between the data, predicting a value as a function of the inputs' data. Depending on the training phase, we can distinguish various learning techniques: (1) Supervised learning (both inputs and outputs are provided to the algorithm), (2) unsupervised learning (only inputs are provided), and (3) reinforcement learning (an objective reinforcement function is maximized). During the inference, the inputs' data pass through the layers and each layer performs matrix multiplications. The output of the final layer is either a number or a classification output. A deep neural network (DNN) [13] (Figure 2) is an artificial neural network (ANN) with multiple layers between the input and output layers and the operations include linear or nonlinear functions. A special case of DNNs involves the usage of the matrix multiplications with convolutional filter operations, which is common in DNNs that are designed for image and video analysis. This type of models is known as convolutional neural networks (CNNs) [54] and they are used when the numbers of input variables are high. The DNNs designed especially for time series prediction are called recurrent neural networks (RNNs) [55], characterized by having loops in their layer connections to keep state and enable predictions on sequential inputs.

There are many possible choices on how to design a NN model, provided that different hyperparameters of the network bring a different level of accuracy. In particular, a model with high accuracy requires more memory than a model with low accuracy due to the number of parameters. The metric used to measure accuracy depends on the domain in which the ML algorithm is applied. For example, in object detection, the accuracy may be measured by the mean average precision (mAP) [56], which measures how well the predicted object location overlaps with the ground-truth location, averaged across multiple categories of objects.

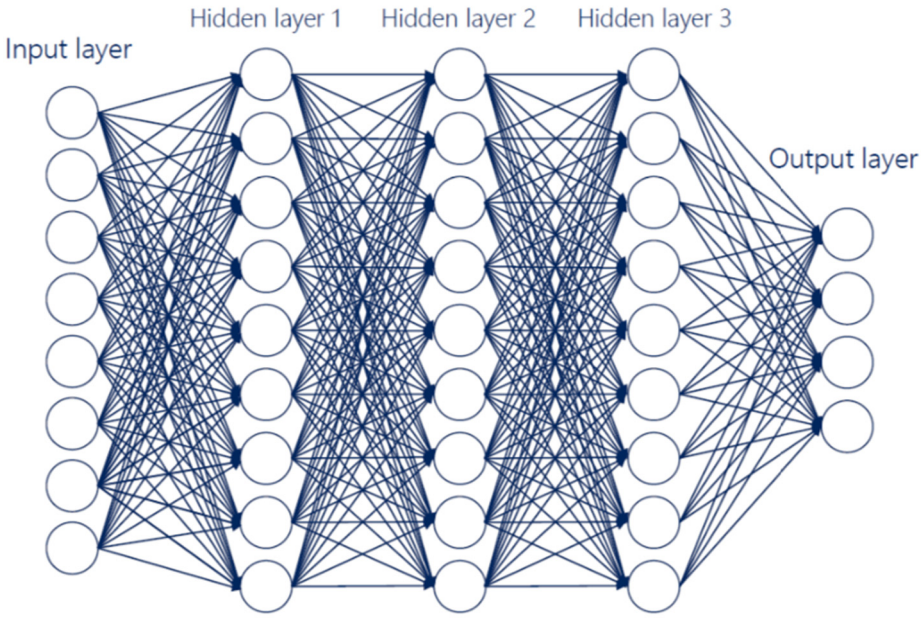

Figure 2. Deep Neural Network (DNN) example.

2.2. RNN, GAN, K-NN

A particular type of NN are the RNNs (recurrent neural networks) [57]. In this type of NN, the output values of a high layer are used as input for a lower one. This interconnection allows the use of one of the layers as state memory. Providing a temporal sequence of values as input, it allows us to model also dynamic temporal behavior. This makes them applicable to predictive analysis tasks on data sequences, such as handwriting recognition or speech recognition [58]. A particular RNN is LSTM (long short-term memory) [59]. A LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate. The cell remembers the values over the time and the gates regulate the flow of information into and out of the cell. In particular, the forget gate can learn what information is kept or forgotten during training.

Another type of NN is the generative adversarial network (GAN) [60]. They consist of two networks: Generator and discriminator. The first generates data after it learns the data distribution from a training dataset of real data. The second one is in charge of classifying the real data from the fake ones generated by the generator.

K-nearest neighbors algorithm (K-NN) [61] is an algorithm used in the field of patterns recognition, based on the characteristics of the objects close to the one considered. This method is used both for classification and regression problems. There are different modified versions of k-NN that helps to implement the algorithm in hardware-constrained devices and the most innovative is ProtoNN [62]. It is a k-NN-based algorithm. The main problems of the K-NN for the computation at the edge are: The training data size (the algorithm generates prediction using the entire datasets), the prediction time, and the choice of the distance metric. To address these issues, ProtoNN works on a smaller training dataset excluding the unnecessary data. The dataset is then projected to a low dimension matrix and jointly learned across all data points. Gupta et al. implemented ProtoNN on an Arduino Uno to evaluate its performance using 14 datasets and reported almost the same classification accuracy as the state of the art.

2.3. Tree-Based ML Algorithms

Tree-based ML algorithms are used for classification and regression problems that are a very common practice in the IoT field. However, due to the limited resources of the devices, the usual tree algorithms could not be brought on them. An emerging algorithm is Bonsai [63]. The tree algorithm [64] is designed specifically for severely resource-constrained IoT devices and it maintains prediction

accuracy while minimizing model size and prediction costs. It learns first a single sparse tree reducing the size model, then it makes nonlinear predictions through the internal nodes and the leaf ones. Eventually, Bonsai learns sparse matrix, projecting all data into a low dimensional space in which the tree is learned. This allows the algorithm to be brought on tiny devices like IoT ones.

The referenced implementation was carried out on an Arduino board populated with an 8-bit ATmega328P microcontroller with 16 MHz operating frequency, 2 kB of Static Random Access Memory (SRAM), and 32 kB of read-only flash memory and on BBC Micro:Bit which has an ARM architecture 32-bit Cortex with an operating frequency of 16 MHz, 16 kB of SRAM and 256 kB of flash.

2.4. SVM

One of the most widely used ML algorithms at the embedded level is the SVM [28,29,43,52]. SVM is a supervised learning algorithm that can be used for both classification and regression problems. The algorithm discriminates between two or more classes of data by defining an optimal hyperplane that separates all classes (Figure 3a). The support vectors are the data closest to the hyperplane, which, if removed, would result in a redefinition of the hyperplane itself. For these reasons they are considered the critical elements of the dataset. Usually, the loss function used by the algorithm is the Hinge loss and the optimization function is the descending gradient technique.

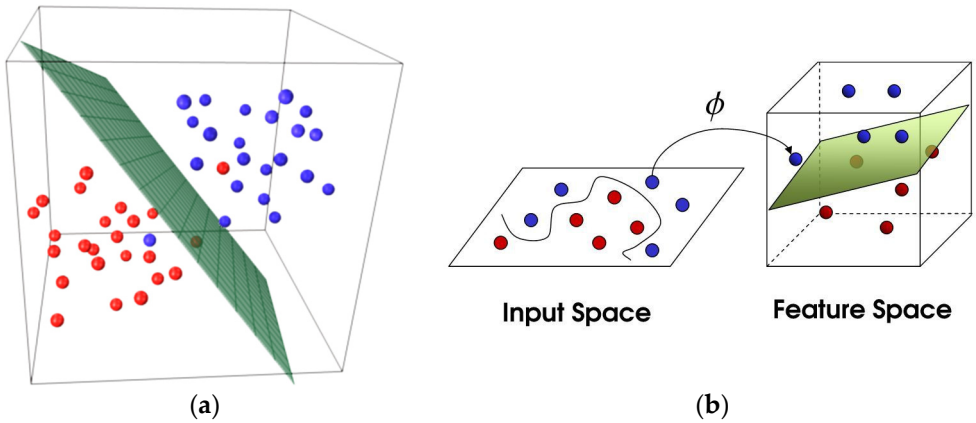

Figure 3. (**a**) Hyperplane that separate two classes of data, (**b**) kernel trick.

Sometimes the data are linearly separable, but this only represents a subset of cases. SVM can efficiently perform a classification using the kernel trick. Suppose we face the problem represented in Figure 3b: It is impossible to find a single line to separate the two classes in the input space. But, after projecting the data into a higher dimension, it is possible to find the hyperplane which classifies the data. Kernel helps to find a hyperplane in the higher dimensional space without increasing the computational cost too much.

3. Bringing Machine Learning to the Edge

3.1. Architectures

To meet latency requirements, different architectures for quick-performing model inference have been proposed. The research focused on three important architectures (depicted in Figure 4): (1) On-device computation, where DNNs are executed on the end device; (2) edge server-based architectures (the data are sent from the end devices to edge servers for computation); and (3) joint computation which includes the possibility to have cloud processing.

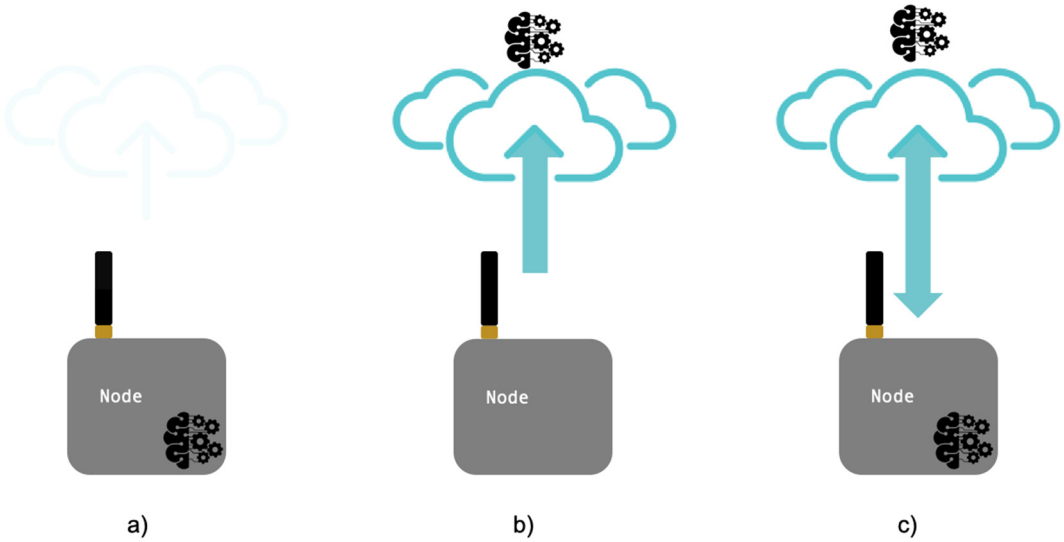

Figure 4. (**a**) On-device computation, (**b**) edge server-based architectures, and (**c**) joint computation.

3.2. Model and Hardware

Several research papers focused on the possibility of bringing artificial intelligence to devices with limited resources [44,65–67] and there have been efforts in decreasing the model's inference time on the device. To bring an AI model on embedded devices, ML developers should deal with the proper hardware choice that fits model design and compression.

3.2.1. Model Design

ML developers focus on designing models with a reduced number of parameters in the DNN model, thus reducing memory and execution latency, while aiming to preserve high accuracy. There are several efficient models designed specifically to execute a NN on devices with low computational capacity and power, such as MobileNets [68] or SqueezeNet [69], originally born for computer vision tasks. MobileNets are based on a streamlined architecture that uses depth-wise separable convolutions to build light-weight deep NNs. SqueezeNet downsamples the data using special 1 × 1 convolution filters.

3.2.2. Model Compression

The model compression allows us to run the model on tiny devices [70] and there are two main ways to reduce the network: Lower precision (fewer bits per weight) and fewer weights (pruning). Post-training quantization reduces computing power demand and energy consumption at the expense of a slight loss in accuracy. By default, the model weights are float32 type variables, which lead to two problems: Firstly, the model is very large because 4 bytes are associated at each weight, with a considerable memory requirement; secondly, the execution is remarkably slow compared to uint8 type variables. It is possible to considerably reduce the weights from 32 bits to 8 bits, obtaining a 4x reduction in the size of the NN. Note that post-quantization is a technique that is carried out after training the model, but it could be done even before training. ML libraries, such as Tensorflow [71] or Keras [72], give the possibility to apply quantization. As stated above, the reduction of the model size can be obtained not only with quantization, but also with pruning techniques that allow the elimination of connections that are not useful to the NN (Figure 5); this leads to a decrease of the computation request and program memory. Quantization and pruning approaches have been considered individually as well as jointly [70]. These two techniques are the basis of NN compression, from which further techniques have been developed. DeepIoT [73] presents a pruning method for commonly used deep learning structures in IoT devices, and the pruned DNN can be immediately deployed on edge devices without modification. Loss-approximating Taylor expansion was used in [74] as a gradient-based importance metric used for pruning. Anwar et al. [75] selected pruning

candidate by hundreds of random evaluations. Yang et al. [76] selected pruning candidate weighted by energy consumption. Early pruning [77] and dynamic pruning [78] explored how to integrate pruning with a better retraining, and saved the retraining time. A technique that is not among those of pruning and quantization, but which has a significant value, is the knowledge distillation. This involves creating a smaller DNN that imitates the behavior of a larger one [79]. This is done by training the smaller DNN using the output predictions produced from the larger one and the smaller DNN approximates the function learned by the larger one.

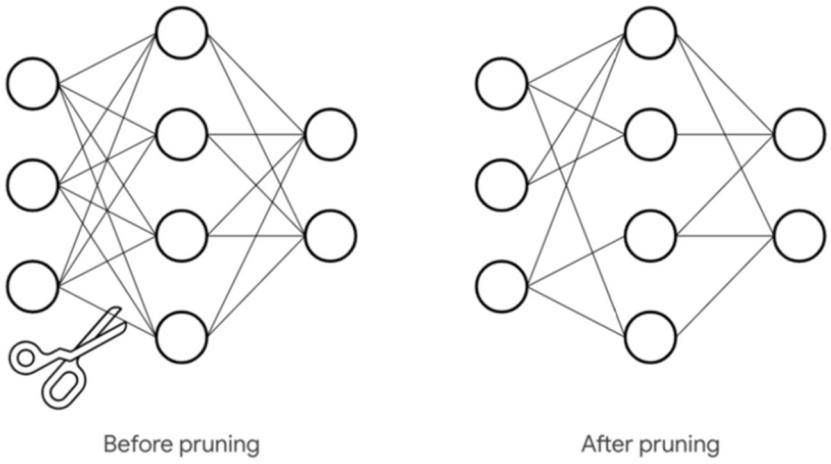

Figure 5. Pruning effect on the network.

3.2.3. Hardware Choice

The choice of the algorithm to be used is important to run a model on an edge device. However, this must also be coupled to an optimal choice of hardware. The metric to be used for choosing the hardware is based on accuracy, energy consumption, throughput, and cost [67]. The accuracy of ML algorithms must be measured on a dataset large enough to be able to affirm that the obtained result is valid. Energy efficiency, on the other hand, is closely related to the programmability and size of the NN. By "programmability" we mean the adaptation of the model to the variation of the context, i.e., the model varies the weights as the scenario varies. By "NN size" we mean, instead, the number of layers that the processor must support. The high size and the variability of the scenario imply an increase in terms of computation. In particular, the high size of the NN increases the number of data, and, instead, the programmability involves the need to access the memory, read the weight value, and modify it. This generally involves an increase in energy consumption. By "throughput" we mean the number of operations required in the unit of time and with "cost" the amount of memory required.

Microcontrollers can be used for AI but implementing the algorithm on them is challenging. They are excellent choices in IoT applications and may run networks that are not too large for low-data fusion tasks. A good tool to facilitate the implementation of a DNN on a microcontroller is the X-CUBE-AI [80], suitable only for STMicroelectronics MCUs. It is an expansion of the STM32CubeMX environment that extends the potential of the tool, allowing an automatic conversion of pretrained NNs to low-resource hardware. X-CUBE-AI also optimizes libraries by modifying layers and reducing the number of weights to make the network more "memory-friendly".

Tiny hardware that can be used for IoT purposes and recommended on the Tensorflow lite website [81] are:

- Arduino Nano 33 BLE Sense [82]
- SparkFun Edge [83]
- STM32 microcontrollers [84]
- Adafruit EdgeBadge [85]

- Espressif ESP32-DevKitC [86]
- Espressif ESP-EYE [87]

Arm has provided its own solution for the IoT field [88]. Recently [89], they announced the introduction of new features on their AI platform, among which new ML Intellectual Property, the Arm® Cortex®-M55 processor that can be up to 15 times faster than the previous version, and Arm Ethos™-U55 NPU, the first micro-NPU (neural processing unit) for Cortex-M architecture, which can speed up ML performance by up to 480 times.

A selection of hardware used for IoT devices that implement edge computing is reported in Table 1.

Table 1. Hardware used for Internet of Things (IoT) devices that implement edge computing.

Work	DNN Model	Application	End Devices	Key Metrics
This work (Section 9)	CNN	Image Recognition	STM32F401RE (ARM® Cortex® -M4)	fast inference
[23]	SVM	Image Recognition	Raspberry Pi model 3 (ARM® v8)	fast inference
[90]	DNN	Distributed Computing	Raspberry Pi model 3 (ARM® v8)	hierarchical
[91]	SVM, CNN	Video Analysis	Raspberry Pi model 3 (ARM® v8)	fast inference
[92]	SVM	Video Analysis	Raspberry Pi model 3 (ARM® v8)	fast inference
[28]	SVM	Battery Lifetime Estimation	SPHERE	energy
[44]	CNN	Image Recognition, Sensor Fusion	Motorola 68HC11	fast inference
[65]	SVM	Code execution	ARM® v7	accuracy
[93,94]	Logistic Regression	Human Activity Recognition	ESP32	accuracy
[95]	CNN	Speech Recognition	Sparkfun Edge	accuracy

4. Edge Server-Based Architectures

The solutions described in the previous sections allow us to run the AI algorithm on end devices but implementing powerful DNNs on tiny devices is still challenging (e.g., decision making and real-time execution). In some circumstances, it is necessary to transfer the computations from end devices to more powerful entities [96]. Since the edge server is close to users, it could be the best approach to solve the problems related to the calculation in optimal times. The easiest way to utilize the edge server is to offload all the computation from end devices to the edge server: The end devices will send their data to a nearby edge server and receive the corresponding results after server processing. When sending data to an edge server, data preprocessing is useful to reduce redundancy and thus decrease communication time. An example could be Glimpse [97] that is a continuous, real-time, object recognition system for camera-equipped mobile devices. Starting from the video, Glimpse identifies objects and labels and traces them. Because the algorithms for object recognition entail significant computation, Glimpse runs almost always on server machines. It uses change detection to filter which camera frames are offloaded. If no changes are detected, Glimpse will perform frame tracking locally on the end device. This preprocessing makes real-time object recognition possible.

Obviously, if the data processing is outsourced to an edge server, more edge devices will belong to it. This should bring the problem of the shared resources. So, the developer should look for the right trade-offs between accuracy, latency, and other performance metrics, such as a number of requests served. A practical solution [98] could be to assign the computation across a hierarchy of edge and cloud servers jointly tuning all the DNN hyperparameters. Mainstream [99] considers a

similar scenario but the proposed solution uses transfer learning to reduce the computational resources consumed by each request. Transfer learning enables multiple applications to share the common lower layers of the DNN model and computes higher layers unique to the specific application, thus reducing the overall amount of computation.

5. Wireless Standards for AI-Enabled IoT Devices

Most of the energy in the IoT device is wasted due to the communication protocol. Indeed, any unnecessary data that are transferred, stored, and processed appear to be a potential waste of energy. Consequently, excellent algorithms must also be accompanied by efficient communication protocols. According to the specific scenario, the developers can use different communication protocols (Table 2). This is because we can distinguish protocols that allow us to transmit a small amount of data over long distances with a low energy consumption and protocols that can transmit a great amount of data over long distances with a high consumption. If the spectrum band use is considered, we can also classify them into technologies that use the licensed or the unlicensed spectrum, e.g., Industrial, Scientific and Medical (ISM) bands [100]. Among the many communication technologies, we include BLE [101], an example [102] that presents the design and optimization of a smart sensor supplied by 2.4 GHz Radio Frequency (RF) power and performing infrared-based motion detection and BLE communication. Bluetooth wireless technology is widely used, including the introduction of Bluetooth 5 [103,104] that uses less power and supports mesh topology, enables large-scale device networks, and many-to-many communications. Bluetooth 5 meets the requirements for recent IoT devices with its good range, increased speed up to 2 Mbps, and a long-range mode with higher sensitivity at lower bit rates.

ZigBee [105] is one of the main IoT communication standards. It is based on IEEE 802.15.4 [106] standard for WPAN (wireless personal area network) and its primary application is in the field of wireless sensor networks (smart energy and home automation) [107]. It could be used in different fields: Smart cities [108,109], agriculture [110,111], automotive [112,113], and health care [114,115]. ZigBee operates mainly in the 2.4 GHz, but also supports the 868 MHz and 916 MHz ISM bands.

Another solution is Z-Wave, a sub-GHz mesh network protocol often used for security systems, home automation, and lighting controls [116]. Like Zigbee, Z-Wave is a low-power technology based on IEEE 802.15.4 that transfers small amounts of data over short and medium distances. Z-wave uses a proprietary radio system and has a strictly regulated product ecosystem targeting smart homes, while Zigbee devices can be used for a variety of applications and are not fully interoperable.

ANT [117] is a proprietary protocol operating in the 2.4 GHz band designed for low bit rate and low-power networks. It supports point-to-point, star, tree, and mesh networks and up to 65,533 nodes for each of the available channels. It was originally used in sports and fitness sensors but later used for home automation and industrial applications. ANT+ is a standardized layer on top of the ANT protocol allowing devices' interoperability [118].

In the wake of the market demands of direct IP-based connectivity, new wireless mesh networking standards have been developed. The 6LoWPAN (IPv6 over low-power wireless personal area networks) [119] is a light-weight, IP-based communication and is an open IoT network protocol, primarily used for home and building automation. However, the standard only defines an efficient adaptation layer between the 802.15.4 data link layer and the TCP/IP stack. Thread [120] is a secure and reliable mesh protocol for home automation running over 6LoWPAN and IEEE 802.15.4 radio. The stack is an open standard built as a collection of existing standards and is optimized for low-power operation, but the application layer is not standardized.

The aforementioned WPAN solutions require an application-level gateway that runs the TCP/IP stack via Ethernet or WiFi. Instead, 6LoWPAN-based solutions use an edge router that only forwards packets at the network layer and does not implement an application layer state, allowing low-cost bridging to other IP networks.

WiFi networks (IEEE 802.11) use an access point (AP) as an Internet gateway and have good data capacity and coverage inside buildings. Until recently it was quite expensive to integrate WiFi connectivity into devices with low computing performance, due to the size and complexity of WiFi

and TCP/IP software and the high power consumption that make it not suitable for use with battery-powered devices. Now, however, new devices support WiFi and TCP/IP software and have reduced power consumption. The power consumption of these devices can be further reduced by activating the radio section only for short periods, allowing them to operate for over a year with two AA batteries [121].

Many different WiFi protocols are available and operate at either 2.4 GHz or 5 GHz. IEEE 802.11n and IEEE 802.11ac are the most widely used protocols but different versions have been developed in the past years for higher versatility. WiFi HaLow (IEEE 802.11ah) is designed for low data rate and long-range devices [122]. It operates in the sub-GHz ISM band and implements power-saving techniques, such as target wake time (TWT) that wake up the device at defined intervals for a very short time. HaLow was released in 2016, but is not yet widely used in commercial products. The 802.11af [123] has the same target applications of HaLow but it relies on unused TV spectrums in UHF and VHF bands and never took off.

The 802.11ax [124] is a more recent version of WiFi technologies that support higher transfer speed and also introduce power-saving features such as TWT, making it more attractive for IoT applications. It also includes features that allow it to extend the range and allows the partition of the channel into smaller subchannels to reduce the data rates while extending the number of devices that can be reliably connected to an access point, allowing it to scale up to thousands of devices.

Radio-frequency identification (RFID) is a technology that uses sub-GHz ISM bands, designed specifically so devices without batteries could send a signal [125]. NFC (near-field communication) is a protocol used for very close communication [126]. It operates in the 13.56 MHz and is designed to exchange data with another NFC device, allowing bidirectional communications. The low data rate and short communication distance make it suitable only for niche IoT applications.

LoRa (long range) [127,128] is a low-power, wide-area network (LPWAN) technology. It is based on spread spectrum modulation techniques and it could be used for empowering the IoT scenario [129]. In [130], it is presented a solution of machine learning on edge devices with the use of LoRa as a low-power transmission protocol. Implementing machine learning with LoRa allow it to reduce transmitted data by 512 times and extend battery life by 3 times for that specific scenario. Nowadays, the most common strategy for processing data is the use of the cloud, but the transmission of large amounts of data requires frequent recharging of the devices, thus negating the prerogatives of the IoT. In addition, IoT applications could require long-distance data transmission, such as for traffic monitoring. IoT devices must, therefore, have a low-energy profile and sometimes be able to transmit over great distances for a given scenario [125,130,131]. Furthermore, IoT devices require edge processing for bandwidth, latency, and privacy issues. Under these conditions, the efficient use of data reduction and local processing must be coupled with long-range and small-bandwidth transmission protocols and this could be obtained using LoRa.

SigFox [132] is another LPWAN solution. It is a narrowband technology and allows the use of simpler devices that are available from different manufacturers. However, it requires the use of sophisticated and expensive gateways and access points and the network is controlled by SigFox and has a fee. On the contrary, LoRa is open and its use is free with no subscriptions and no constraints on installation of gateways and network servers. However, the production of LoRa radio is a Semtech exclusive.

LTE (long-term evolution), commonly known as "4G LTE", is a standard for wireless broadband communication based on GSM/EDGE and UMTS/HSPA technologies [133]. After the LTE introduction, it began to compete with the emerging technologies for IoT field such as BLE, narrowband Internet of Things (NB-IoT), ZigBee, and LoRa. The 4G has improved the capabilities of cellular networks but it is not fully optimized for IoT applications [134,135].

NB-IoT [136], has been introduced to provide low-cost, low-power, wide-area cellular connectivity for the Internet of Things. NB-IoT is a standards-based low-power wide-area (LPWA) technology developed to enable a wide range of new IoT devices and services. NB-IoT significantly improves the power consumption of user devices, system capacity, and spectrum efficiency, especially in deep coverage. NB-IoT is built [137] from existing LTE functionalities with essential

simplifications and optimizations. At the physical layer, it occupies 180 kHz of spectrum, which is substantially smaller than LTE bandwidths of 1.4–20 MHz. At the higher layers, simplified LTE network functions are supported. Compared to other LPWAN solutions, NB-IoT has the great advantage of eliminating the need for a specific gateway, so sensor data is sent directly to the cloud server, reducing infrastructure costs.

LTE Cat-M1 [138] is a LPWAN that enables cellular services for the IoT world. Compared to NB-IoT, this technology provides higher data rate and the ability to use voice over the network, but requires more bandwidth and, therefore, the devices are more complex and expensive. NB-IoT and Cat-M1 have different and somewhat complementary target applications with the former suitable for small sensors and meters and the latter for devices that require higher data rates and have a higher power budget.

The 5G networks and standards are expected to solve challenges that are facing 4G networks. The 5G is the fifth generation of mobile, cellular technologies, networks, and solutions. Although not just 'built' for the Internet of Things (IoT), it will be the major driver of the growth of IoT. The 5G IoT is a novel [139], intelligent network based on 5G communication, which is designed to connect sensing regions (sensors) and processing center (cloud) provided by AI algorithms. It presents the different emerging technologies, involving massive Multiple Input Multiple Output (MIMO) networks, dense static small-cell networks, and mobile small-cell networks. The 5G fulfills the needs of the IoT [135]:

- high data rate;
- high scalable and fine-grained networks, to increase network scalability;
- very low latency;
- long battery lifetime, to support billions of low-power and low-cost IoT devices.

Reducing the latency in the communications, 5G eliminates part of the bottleneck related to the remote execution of ML algorithms [140].

ML developers should properly define the communication technology based on specific design requirements, as well as architectures, hardware, latency, and strategy of computation [141–143].

Table 2. Main communication technologies used in IoT.

Group	Technology	Data Rate	Distance (Indoor/Outdoor)	Works
Contactless	NFC	424 kbps	0–4 cm	[126]
Contactless	RFID	640 kbps	10–20 m	[125]
LPWAN	LoRa	0.3 to 50 kbps	5–10 km	[127,128,144–148]
LPWAN	SigFox	100 or 600 bps	30–50km	[143,148–151]
WPAN	Zigbee	250 kbps	10–100 m	[152–155]
WPAN	Z-Wave	100 kbps	100 m	[116,156]
WPAN	Bluetooth LE	1 Mbps	10 m/50 m	[102,157–159]
WPAN	Bluetooth 5	2 Mbps	40 m/200 m	[160–162]
WPAN	ANT	60 kbps	30 m	[163]
WiFi	IEEE 802.11n	600 Mbps	70 m/250 m	[164]
WiFi	IEEE 802.11ax	9600 Mbps	30 m/120 m	[124]
WiFi	IEEE 802.11af	570 Mbps	280 m/1 km	[165,166]
WiFi	IEEE 802.11ah	347 Mbps	140 m/500 m	[122,166,167]
Cellular	NB-IoT	200 kbps	280 m/1 km	[136,137,150,168]
Cellular	LTE-M1	1 Mbps	5–100 km	[138]
Cellular	4G/LTE	150 Mbps	15 km	[169]
Cellular	5G	10–50 Gbps	2 km	[170–172]

6. Joint Computation

Although the edge server can accelerate DNN processing, it is not always necessary to have the edge devices executing DNNs on edge servers. We will introduce three offloading scenarios (Figure 6): (1) partial offloading of partitioned DNN (the decision is what fraction of the DNN computations should be offloaded), (2) hierarchical architectures (offloading is performed across a combination of edge devices, edge servers, and cloud), and (3) distributed computing approaches (the DNN computation is distributed across multiple peer devices).

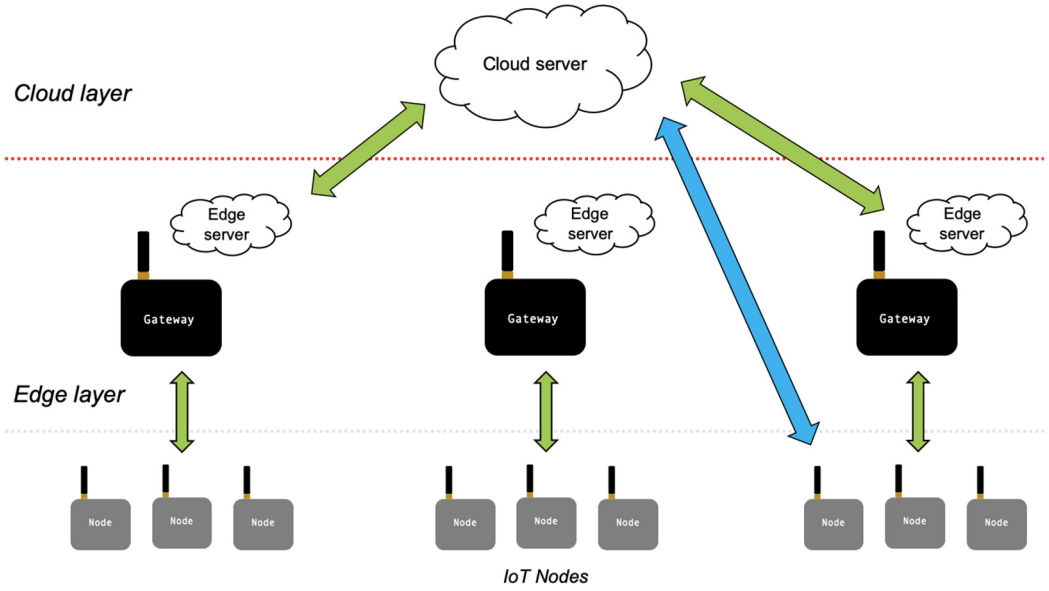

Figure 6. Joint computation among devices, edge, and cloud servers.

6.1. Partial Offload

In model partitioning approaches, some layers are computed on the device and the others are computed by the edge server or the cloud. This approach can potentially offer latency reductions thanks to the compute cycles of other edge devices. Indeed, after the first few layers of the DNN model have been computed, the size of the intermediate results is relatively small and the output can be sent over the network to an edge server in a faster way than the original raw data [90]. Critical is the choice of the point where the network needs to be partitioned and one algorithm that can be used is Neurosurgeon [173]. It is a light-weight scheduler used to automatically partition DNN computation between mobile devices and datacenters at the granularity of NN layers. So, it decides where to partition the DNN, layer-wise, while accounting for network conditions.

The partition could also be applied to the input data (e.g., raw image) and this is useful for hardware with constrained memory that is largely used in the IoT scenario, such as IoT sensors. However, input-wise partitioning can result in increased data dependence, as computing subsequent DNN layers requires data results from adjacent partitions. DeepThings [90] uses input-wise partitioning.

6.2. Hierarchical Architectures

ML algorithm can be performed on edge devices and on the cloud. Entrusting the computational task to the cloud could create a latency problem. Instead, the use of powerful computational cloud resources can potentially decrease the total processing time. For example, Li et al. [45] divided the DNN model into two parts: The edge server computes the initial layers (lower layers) of the DNN model after it received the input data and then the cloud computes the higher layers of the DNN. The cloud sends back the final results to the end devices after processing. In this way, the cloud helps the edge server with the heavier computations. There are also other approaches like DDNN [174] (distributed deep neural networks) in which the computing is distributed across an hierarchical system, consisting of the cloud, the edge (fog), and end devices. DDNN also allows fast and localized

inference using shallow portions of the NN at the edge and end devices. Due to the distributed nature, DDNNs improve the fusion of the data from network sensors, system fault tolerance, and privacy for users. Generally, a common feature for the edge approaches is that the edge server serves a limited geographical area, so the input data and, thus, their DNN outputs may be similar.

6.3. Distributed Computing

Hierarchical scenario is based on the offload of the network to more powerful entities like edge devices or cloud. In the distributed perspective the DNN computations can be distributed across multiple peer edge devices, like in DeepThings [90]. It distributes the DNN executions between end devices such as Raspberry Pi and Android smartphones. The DNN partition choice is based on the computation capabilities and memory of the end devices.

7. Privacy

Both in edge server-based architectures and in joint computation, the data are exchanged over the network (e.g., from end device to edge server or from edge server to cloud) and it may contain sensitive information. This can lead to privacy issues. In fact, as already mentioned, edge servers work locally in a geographically limited area. Therefore, the origin of the data is practically known. Although ML on edge devices allows the data reduction on the network and, therefore, improving privacy by itself, it is possible to improve the system through additional techniques, such as adding noise to data or cryptographic techniques.

7.1. Add Noise to Data

A solution is to add noise to the samples uploaded on the network during inference. Wang et al. [175] deployed a smaller DNN locally on the edge device to extract features, add noise to the features, and then upload the features to the cloud for further inference processing by a more powerful DNN. The DNN on the cloud is pretrained with noisy samples so that the noisy inference samples uploaded from the end devices can still be classified with high accuracy at test time. This is based on differential privacy mechanism [176–179].

7.2. Cryptographic Techniques

Cryptographic techniques can be used to compute the inference with a high level of privacy. The target of secure computation [66] is to ensure that the end device receives an inference result without learning anything about the DNN model on the edge server and vice versa. One method of secure computation is homomorphic encryption, in which the communicated data are encrypted and computation can be performed on the encrypted data, as done in CryptoNets [180–182]. The DNN is converted in CryptoNets, approximating the common activation functions and operations in a DNN, in a low-degree polynomial, which guarantees the homomorphic encryption. However, a bottleneck of the homomorphic encryption tends to be its compute times. Multiparty computation is another technique for secure computation. In secure multiparty computation, multiple machines work together and communicate in multiple rounds to jointly compute a result. Secure multiparty computation focuses on the privacy of the intermediate computation steps, but its bottleneck tends to be the communication complexity.

8. Training

Thus far, edge computing and deep learning have mostly been discussed assuming that a deep learning model has already been trained offline on a prebuilt dataset. This section presents a discussion on training algorithms and hardware for the edge field. Usually, training data produced by end devices would be sent to the cloud, which would then perform the training and finally distribute the trained model back to the edge devices. Leaving data at the edge is useful when privacy is highly desired and also helps to reduce the network bandwidth requirements.

8.1. Training Algorithms

Exchanging model parameters and other data between edge devices and cloud servers is mandatory for training an edge–cloud-based DL model. However, as the size of the training model increases, more data needs to be exchanged between edge devices and servers. The high network communication cost is a bottleneck for a training model, and a local edge training implementation is required. An example of local networks is a mobile computing system for DNN applications (MoDNN). MoDNN [183] uses a pretrained DNN model and scans each layer of the DNN model to identify layer types. If the layer is a convolutional one, the input layer is divided by biased one-dimensional partition (BODP) method. BODP decreases the computing by reducing the input size. If a fully connected layer is detected, the layer input is assigned to different work nodes (mobile devices) to achieve the minimum total execution time. In this case the network does not change the weights as the external scenario varies because it is pretrained. However, the structure of the edge system is hierarchical and the training can be distributed among peer edge devices and the cloud. In-edge AI [6] is a framework which allows better collaboration among devices and edge nodes to exchange the learning parameters for a better training and inference of the model. It integrates deep reinforcement learning techniques and federated learning for mobile computing purpose. Teerapittayanon et al. [174] used a cloud server for training the DDNN among different devices (including edge devices and the cloud), while the most powerful one trains the network. The training of DDNNs is difficult because of multiple exit points. To address this issue, the network was trained jointly by combining losses from each exit point during back-propagation. The training could be made also on pruned model: Chandakkar et al. [184] designed a new architecture to retrain a pruned network on an edge device. A complete DNN is trained for an epoch (when an entire dataset is passed forward and backward through the DNN) on the original data. Also, layer-wise, magnitude-based weight pruning is performed with a user-defined threshold value. This approach greatly reduces the computational complexity by removing connections in a DNN model and makes it suitable to run on a limited resources device. Unfortunately, any pruning process reduces the accuracy of a model. To overcome this issue, this approach finds the indices of the most important weights for an important feature and excludes these elements from being pruned. Finally, the pruned DNN network is used while training the next epoch because these operations are performed cyclically.

To reduce communication costs and keep model accuracy high, Tao and Li introduced a new method called edge stochastic gradient descent (eSGD) [185]. In this approach, all edge devices run training tasks separately with independent data and the gradient values generated by the edge devices are sent to the cloud servers. The server, after obtaining the gradients from the end devices, uniforms the gradients by performing the average. After that, it updates the parameters by using this average value. These updated parameters are sent back to the edge devices for the next training step. This process is called parameter synchronization. Unfortunately, this gradient selection technique decreases model accuracy. The eSGD uses two mechanisms to maintain a high level of accuracy for the training:

- 'Important' updating: After each mini-batch, only a small fraction of the gradient coordinates need to be updated. The algorithm determines main gradients, which will then be updated by the server. This process significantly reduces communication cost.
- Momentum residual accumulation: This mechanism is applied for tracking and accumulating out-of-date residual gradients, which helps to avoid low convergence rate caused by the previous important updating method.

The eSGD is capable of reducing the gradient size of a CNN model by up to 90%. Unfortunately, high gradient shrinking leads to bad accuracy. Tao and Li used Modified National Institute of Standards and Technology (MNIST) database in their experiments and reported 91.22% accuracy with a 50% gradient drop.

8.2. Training Hardware

Updating the neural network or computing complex algorithms cannot be completely entrusted to tiny hardware like microcontrollers. Field-programmable gate array (FPGA) and graphical processing unit (GPU) consume too much power (the FPGA is still a better choice than the GPU since it is versatile and consumes less power), but they turn out to be excellent for training NNs or performing powerful algorithms [186]. GPUs use temporal architectures such as SIMD (single instruction multiple data) or SIMT (single instruction multiple threads) to perform the MACs in parallel and there are software libraries designed for GPUs that optimize the matrix multiplications e.g., NVIDIA CUDA® Basic Linear Algebra Subprograms (cuBLAS) [187], NVIDIA CUDA® Deep Neural Network library (cuDNN) [188]. The matrix multiplications on these platforms can be further improved by applying transforms to the data to reduce the number of multiplications. Fast Fourier transform (FFT) [189] is a well-known approach that reduces the number of multiplications from $O((N^2_o)(N^2_f))$ to $O(N^2_o log_2 N_o)$, where the output size is $N_o * N_o$ and the filter size is $N_f * N_f$; however, the benefits of FFTs decrease with filter size. Other approaches include Strassen [190] and Winograd [191].

Recently, some very interesting devices are emerging, such as the Hailo-8 DL [192] from the Hailo company. The Hailo-8 DL is a processor suitable for performing deep-learning at high levels and allows for very high performance on end devices with minimum power consumption, size, and costs. In particular, it offers high performance (26 tera-operations per second) and is very efficient and highly flexible (reprogrammable). Google has developed an ASIC (application-specific integrated circuit) dedicated to the TensorFlow library TPU (Tensor processing unit) [193], whose computational capacity is 180 teraflops. These are examples of IA accelerators, such as NPU, that is a class of microprocessors designed to provide hardware acceleration to artificial NNs, automatic vision, and ML algorithms for robotics, IoT, and other data-based applications [194]. While hardware DNN accelerators are quite new, there have already been two branches of designs. The first class of accelerators only looked at the data flow, ignoring the memory energy consumption. The second one tried to address the amount of energy consumption due to memory access. The first style of accelerators include ConvNet Processor (CNP) [195], Neuflow [196], and dynamically configurable (DC) CNN [197], proposing customized logic to map convolution to hardware with more parallelism and flexibility. The second wave of accelerators focused on optimizing memory transfer and data movement. As modern NNs get larger, researchers realize that memory access and moving data is more critical than matrix products between layers. Among these accelerators (TPU is included in this class of accelerators), DianNao [198] implements an array of multiply-add units to map large DNNs onto its core architecture. It has customized on-chip buffer to minimize Dynamic Random Access Memory (DRAM) traffic. DaDianNao [199] and ShiDianNao [200] eliminate the DRAM access by having all weights on-chip. An interesting AI accelerator is Movidius stick [201] suited for edge computing because it makes easy to add deep learning capabilities to existing computing platforms. It is designed mainly for computer vision tasks at the edge [202] and allows deploying CNNs on low-power applications that require real-time inferencing. A detailed guide on the use is reported in [203].

Another AI accelerator is Coral [204]. It is a platform from Google that allows realizing devices with local AI, providing hardware acceleration for neural networks at the edge of the network without any help from the clouds. At the base of Coral there is Google's Edge TPU, an ASIC chip optimized to run lightweight machine learning algorithms. Many applications are reported by Coral project itself [205]. A selection of AI accelerator devices that implement edge computing is reported in Table 3.

Table 3. Artificial Intelligence (AI) accelerator devices that implement edge computing.

Work	DNN Model	Application	End Devices
[206–208]	SVM/CNN	Image and Video Analysis	Movidius
[209–211]	CNN	Image and Video Analysis, Robotics	Jetson TX1
[212,213]	YOLO [214]	Image Recognition, Robotics	Jetson TX2
[98]	AlexNet	Image Classification	Nvidia Tegra K1
[196]	CNN	Image Analysis	Neuflow

[215]	CNN, DNN	Image Recognition	DianNao
[200]	CNN	Vision Processing	ShiDianNao

9. MNIST Example

In this section we will analyze how it is possible to port a NN to the embedded environment using one of the most famous models in machine learning world: the MNIST, the ML "Hello World". We will show how to create the neural network using the Tensorflow library and subsequently Keras, using techniques such as pruning and quantization to reduce the size of the model. Finally, the algorithm will be implemented on the NUCLEO-F746ZG board through the X-CUBE-AI tool.

9.1. Dataset

The dataset MNIST was developed by Yann LeCun [216], Director of the Facebook Research Center for Artificial Intelligence, to recognize numeric digits. The dataset was created from a series of documents made available by the NIST (National Institute of Standards and Technology) [217] and the images were normalized in size and centered. In particular, the dataset provides 28 × 28 handwritten images with a total of 784 pixels per image, with a splitting of the dataset to implement training and evaluation of the model to overcome the overfitting issue. The training set consists of 60,000 samples and the test set of 10,000 samples. The objective is to write an algorithm that allows recognizing which digit has been written. Since there are 10 types of digits (numbers from 0 to 9), the problem can be seen as clustering task with 10 possible classes. In a first instance, we will show the realization of the NN using a DNN with 2 hidden layers between input and output; then we will show how to implement the same problem using a CNN and Dropout [218] to increase the accuracy of the model.

9.2. Model with Tensorflow

In this first investigation, we will present the implementation of the NN with Tensorflow. The image was saved in a vector of 784 elements in which each element corresponds to the intensity of the color associated to the pixel. With the samples normalized, the values closer to 0 are close to white, and those closer to 1 are classified as black. As already said, this is a classification problem, so the targets are categories. One way to represent the classes is one hot encoding, which is optimal in the case of limited classes: The target for each sample fed to the NN is a vector of length 10 (e.g., if we feed the NN with the digit of value 4, the target associated with it should be [0,0,0,0,1,0,0,0,0,0]).

The NN consists of an input layer (dimensions: 784), two hidden layers (dimensions: 50), and an output layer (dimensions: 10). Since we are working with a DNN, activation functions are mandatory. On the basis of several test carried out, the choice falls on a *relu* for the first layer and a *sigmoid* function for the second layer, as this couple produce the higher level of accuracy (Table 4).

Table 4. Accuracy for different activation functions.

First Level	Second Level	Accuracy on Test
relu	relu	96.20%
tanh	tanh	96.80%
sigmoid	sigmoid	96.96%
relu	tanh	97.18%
tanh	relu	96.64%
sigmoid	relu	96.88%
relu	sigmoid	97.25%
tanh	sigmoid	97.21%
sigmoid	tanh	97.10%

The loss function used, being a classification problem, is the cross entropy applied directly on the *softmax*. The function *tf.nn.softmax_cross_entropy_with_logits(logits, labels)*, combines the two operations

making it faster, but also numerically stable. Instead, as optimization function it is possible to use an adaptive function, i.e., ADAM. Once the model is defined, we have all the requirements for the training of the NN. Before starting the training, it is necessary to initialize the variables (weights and biases) using the *tf.global_variables_initializer()* method. Tensorflow uses Xavier [219] as default initialization. Then, it is necessary to define the size of the batches, their number as a function of the size of the dataset, and a threshold for the loss function related to the validation dataset, so that if the error increases, early stopping prevents overfitting. It should be noted that the threshold has been set at a high value, so that early stopping will not occur at the first time. The final part of the required code is related to the actual training, realized through a cycle. For each period, which is repeated to be a complete iteration of the dataset, a relative to the batches is defined, through which it is possible to calculate the average error relative to the single period as the sum of the errors associated with the batches on the number of batches. At the end of each epoch, it is possible to calculate the loss relative to the validation dataset; if the current validation error is higher than the previous one, then the model has conformed too much to the dataset and, therefore, has no ability to adapt and it is necessary to stop learning using early stopping.

The model made with Tensorflow is too heavy in terms of memory occupation for an edge application; in this example the NN weighed around 15 MB. Tensorflow Lite (TFLite) [220] was created specifically to overcome this problem, proposing a set of tools that help programmers to run embedded, mobile, and IoT devices IA models. In the following, we will show how to use TFLite to bring the NN on a microcontroller. The workflow that we will follow in this tutorial is the following:

- Definition of the model in Keras (using Tensorflow backend),
- Conversion of the model from Keras to TFLite,
- Implementation of a post-training quantization to further decrease the dimension of the NN,
- Design of a Graphical User Interface (GUI) to draw the digit, and
- Test on hardware devices.

9.3. Keras Model

The NN can be built using CNN and the dropout technique. The use of CNN is not necessary, but it is recommended since the number of input variables is very high (CNN allows training the model on a smaller dataset, reducing considerably the number of parameters to learn). The model is defined using the *Sequential()* method which, according to the documentation [221], allows defining the model as a linear stack of layers. The first layer is a 2D convolution layer that creates a convolutional kernel that is superimposed convolutionally with the input layer to produce an output tensor.

To the second layer, also convolutional, is added the *MaxPooling* operation for spatial data (2D). The Dropout is then applied to the network in input. The last two layers are densely connected layers; at the penultimate layer a *relu* is applied as activation function, whereas at the last one a *softmax*. Finally, for the training phase, *crossentropy* is used as loss function and *Adadelta* as optimization function. A summary of the parameters is shown in Table 5.

Table 5. Model outline.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_1 (MaxPooling2)	(None, 12, 12, 64)	0
dropout_1 (Dropout)	(None, 12, 12, 64)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_1 (Dense)	(None, 64)	589888
dropout_2 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 10)	650

Total parameters, 609,354; trainable parameters, 609,354; nontrainable parameters, 0.

After defining the model and parameters for the training phase, it is possible to train the network. With the presented configuration, it is possible to observe that the use of a CNN is able to increase the model accuracy from 96% to 99%.

9.4. Tensorflow Lite

The model in Keras was too heavy for an embedded solution (7172 kB), having embedded devices memories of a few hundred of kB. However, TFlite and the TFliteConverter tool allows considerably scaling the NN down to 2.4 MB. TFlite is characterized by two main components [222]:

- The interpreter runs the optimized models on different hardware types (including mobile phones, low computational capacity devices, and microcontrollers), and
- The converter, which converts the model to a more efficient format for use by the interpreter.

In our case, the converter was used to adapt the model to the TFLite format (serial format is based on FlatBuffers library [223]).

```
model = 'Model_Keras_MNIST_CNN_Test.h5'.
converter = tf.lite.TFLiteConverter.from_keras_model_file(model)
tflite_model = converter.convert()
```

Tensorflow provides tools (Tensorflow Model Optimization Toolkit) for the optimization of the model. The toolkits support techniques used to:

- Reduce latency and inference costs, and
- Implement IA models on edge devices with limited capacity and low-power profile.

These techniques include post-training quantization and pruning techniques. Unfortunately, the quantization of TFLite models is not supported by X-CUBE-AI and, therefore, we selected the NN compression adopted by the ST software.

9.5. Pruning

The reduction of the model can be obtained not only with quantization techniques, but also with pruning techniques that allow eliminating connections not essential for the NN and consequently reduce the number of computations and the demand of memory space for the NN. Also, for this purpose, it is possible to use the libraries provided by TensorFlow and their examples [224]. As discussed in the previous paragraph, the network is redefined importing the *tensorflow_model_optimization* Application Programming Interfaces (APIs). The APIs can be applied either to the single layer or to the whole model. In our example, we applied the APIs to the single layer. The pruning technique consists of iteratively removing connections between layers, given a sparsity parameter (percentage of weights eliminated) and scheduling (pruning frequency). To help the model convergence, connections should not be eliminated immediately but every tot; in this example we set the elimination starting from 2000 step every 100 steps. Next, it is necessary to define, among the pruning parameters, the end step. Then, the model is defined by setting the pruning parameters and applying them to the NN. Finally, it is possible to convert the model and make the quantization. The technique reduces the number of parameters and the computations, preserving the model's accuracy in terms of predictions. The main impact is due to quantization, but also pruning contributes to this purpose by increasing the inference speed and reducing the amount of energy used, thus allowing the use of the IA model on devices with low energy profile and low computational power.

9.6. Graphical User Interface

To test the model validity once brought to the microcontroller, a suitable GUI that allows the user to type the numeric digit can be used, as well as the direct transfer of saved handwritten digit as

input data. The GUI can be made with PIL [225] (Python Imaging Library), Tkinter [226], de facto standard of GUIs in Python. The GUI allows drawing the numeric digit using the mouse motion on the canvas object (Figure 7). Once the drawing is finished, it is possible to export the image as csv, a format supported for the validation on target made by X-CUBE-AI tool.

Figure 7. Digit "6" drawn by the user.

9.7. Validation on Target

The STMicroelectronics NUCLEO-F746ZG board [227] (Figure 8) and the STM32CubeMx tool were used for deploying the model to a real hardware. To guarantee higher performance, the operating frequency of the microcontroller must be set to 216 MHz, and the cache should be enabled. To test the model on the microcontroller, it is necessary to enable the X-CUBE-AI tool by choosing *Validation* as project mode. The Universal Synchronous-Asynchronous Receiver/Transmitter (USART) can be used to let the Personal Computer (PC) communicate with the microcontroller. The artificial intelligence tool [80] is used to bring the model on the device. The tool does not yet support techniques such as quantization for NNs defined with TFlite and pruning, but it is possible to load on the microcontroller the nonquantized model using the compression provided by the program itself. In particular, the compression method aims to optimize memory usage both in terms of Read-only memory (ROM) and RAM, using a dataset-less approach. The reduction of the NN by the tool is made possible through the use of various expedients [80]:

- Weight compression: It is applicable only to dense layers (or fully connected layers) and is based on weight-sharing algorithms such as K-means clustering.
- Layers fusion: It allows merging two layers to optimize data placement, decreasing the number of the DNNs layers (e.g., nonlinearities or pooling after a convolutional layers).
- Activation function optimization: Part of the memory is used to store temporary hidden layers values, so activation memory is reused across different layers.
- Once the model is compressed (in this example we opted for a x4 compression), the tool gives the possibility to make an analysis of the NN to understand if it is loadable on the chosen microcontroller and to visualize the diagram of the loaded model. The Table 6 reports the output analysis of the network implemented in the example. It includes:
- RAM: Indicates the size of the memory required to store the intermediate calculations;
- ROM/Flash: Indicates the memory size needed to store weight and bias after compression; and
- Complexity: Reports the complexity of the model in MAC (multiply-accumulate operations), unit of measure used also to express the complexity of the activation functions.

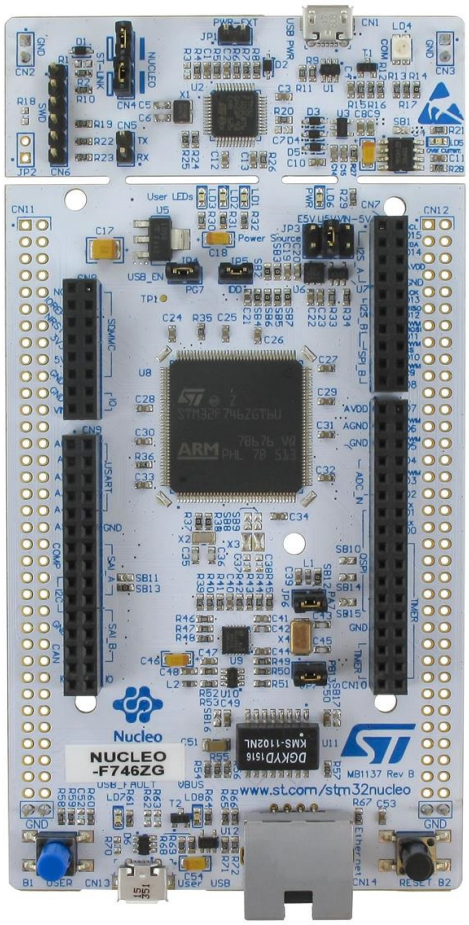

Figure 8. STMicrolectronics NUCLEO-F746ZG.

Table 6. Prediction of model on hardware.

Name	RAM	FLASH	Complexity
Network	135.68 kBytes	668.97 kBytes	11497654 MAC

Finally, it is possible to proceed to the validation to compare the defined model with the one generated in C language by the ST tool, feeding both models the same set of data. The validation can be carried out as:

- Validation on desktop: The model in C is executed on the PC.
- Validation on target: The generated model is executed on the device of interest. It is necessary to load the code on the microcontroller and set a serial communication to communicate with the host.

In both cases the data can be either randomly generated by the tool or can be imported from outside as csv file. After loading the code on the microcontroller, it is possible to enable the validation on the target, and to load the data generated using the GUI as input. The STM32CubeMx reports the model results and it is possible to notice that, in this case, the NN allows effectively recognizing the numeric digit between 10 classes (Figure 9) with an accuracy of 100.00%, root-mean-square error (rmse) = 0.0000, and medium average error (mae) = 0.0000. In Figure 10, the results obtained during validation are reported; the calculation took about 330 ms and the execution time layer by layer is shown in Table 7.

With this example, the full implementation of an eML application of image recognition has been designed and put into practice with good performances.

C0	0									
C1		0								
C2			0							
C3				0						
C4					0					
C5						0				
C6							1			
C7								0		
C8									0	
C9										0

Figure 9. Inference result showing the recognition of the digit "6" drawn by the user (accuracy = 100.00%, root-mean-square error (rmse)= 0.0000, medium average error (mae) = 0.0000, 10 classes, 1 sample).

```
ON-DEVICE STM32 execution ("network", COM7, 115200)..
<Stm32com id-0x25fd4db03c8 - CONNECTED(COM7/115200) devid-0x449/STM32F74xxx msg-2.0>
0x449/STM32F74xxx @216MHz/216MHz (FPU is present) lat=7 Core:IS/DS ART:
found network(s): ('network')
description : 'network' (28, 28, 1)-(6)->(1, 1, 10) macc=11497654 rom=653.29KiE
ram=132.50Ki8
tools versions : rt=14, 0, 0) tool=(4, 0, 0)/(1, 3, 0) api=(1, 1, 0) "Fri Sep
13 18:01:20 2019"
Running with inputs=(1, 28, 28, 1)..   1/1
RUN Stats : batches=1 dur=0.813s tfx=0.766s 4.048Ki15/s (wb=3.062KiE,rb=4015)
Results for 1 inference(s) @216/216MHz (macc:11497654)
duration : 328.458 ms (average)
CPU cycles : 70946870 (average)
cycles/MACC : 6.17 (average for all layers)
```

Figure 10. Inference details.

Table 7. Time contribution of each layer.

Description	Shape	ms
10004/(2D Convolutional)	(26, 26, 32)	9.328
10011/(Merged Conv2d/Pool)	(12, 12, 64)	299.524
10005/(Dense)	(1, 1, 64)	19.562
10009/(Nonlinearity)	(1, 1, 64)	0.006
10005/(Dense)	(1, 1, 10)	0.022
10009/(Nonlinearity)	(1, 1, 10)	0.014
		328.458 (total)

10. Conclusions

Deploying machine learning on Internet of Things devices reduces the network congestion by allowing computations to be performed close to the data sources, preserving privacy in uploading data, and reducing power consumption for continuous wireless transmission to gateways or cloud servers. The aim of this work was to provide a review of the main techniques that guarantee the execution of machine learning models on hardware with low performances in the Internet of Things paradigm, paving the way to the *Internet of Conscious Things*.

In this work, a detailed review on models, architectures, and requirements on solutions that implement edge machine learning on IoT devices was presented, with the main goal to define the state of the art and envisioning development requirements.

The review focused on ML systems deployed on edge devices, providing a comparison between the ML algorithms implementable in edge computing. In addition, the process of bringing ML to the edge was analyzed in detail, considering edge server-based architectures and joint computation, thus envisioning both the case of the absence (and the related effect on *privacy* and *local computational* operations) and the presence (and how it impacts on *cloud/edge server* communications and remote *data transmission* power consumption) of data transmission to gateways or servers.

The actual state of development of edge computing foresees a series of variegated solutions able to satisfy a plurality of needs. Depending on the requirements (privacy, energy consumption, computational complexity), it is possible to define a set of compatible hardware and software to implement AI-enabled IoT effective solutions.

An example of edge machine learning implementation is provided in the review, demonstrating the effectiveness and ease of use of the proper edge-platform used for implementing the machine learning "Hello World".

Author Contributions: Conceptualization, M.M.; methodology, M.M.; software, C.P.; validation, C.P., D.I. and M.M.; formal analysis, M.M. and C.P.; investigation, C.P.; resources, M.M.; writing—original draft preparation, C.P., D.I. and M.M.; writing—review and editing, C.P., D.I. and M.M.; supervision, M.M.; project administration, M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: *Programma di Azione Coesione PAC Calabria 2014–2020, Asse Prioritario 12, Azione 10.5.12*, is gratefully acknowledged by one of the authors (D.I.).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A survey. *Comput. Networks* **2010**, *54*, 2787–2805.
2. Mahdavinejad, M.S.; Rezvan, M.; Barekatain, M.; Adibi, P.; Barnaghi, P.; Sheth, A.P. Machine learning for internet of things data analysis: A survey. *Digit. Commun. Networks* **2018**, *4*, 161–175.
3. IoT: Number of Connected Devices Worldwide 2012–2025 | Statista. Available online: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ (accessed on 21 February 2020).
4. Vahid Dastjerdi, A.; Buyya, R. Fog Computing: Helping the Internet of Things Realize. *IEEE Comput. Soc.* **2016**, *49*, 112–116.
5. Liu, Y.; Yang, C.; Jiang, L.; Xie, S.; Zhang, Y. Intelligent Edge Computing for IoT-Based Energy Management in Smart Cities. *IEEE Netw.* **2019**, *33*, 111–117.
6. Wang, X.; Han, Y.; Wang, C.; Zhao, Q.; Chen, X.; Chen, M. In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Netw.* **2019**, *33*, 156–165.
7. Savaglio, C.; Ganzha, M.; Paprzycki, M.; Bădică, C.; Ivanović, M.; Fortino, G. Agent-based Internet of Things: State-of-the-art and research challenges. *Futur. Gener. Comput. Syst.* **2020**, *102*, 1038–1053.
8. Neto, A.R.; Soares, B.; Barbalho, F.; Santos, L.; Batista, T.; Delicato, F.C.; Pires, P.F. Classifying Smart IoT Devices for Running Machine Learning Algorithms. In *Anais do XLV Seminário Integrado de Software e Hardware*; SBC: Nashville, TN, USA, 2018.
9. Edge Computing—Explore—Google Trends. Available online: https://trends.google.com/trends/explore?date=all&q=edge computing (accessed on 5 March 2020).
10. Scopus Preview - Scopus - Welcome to Scopus. Available online: https://www.scopus.com/ (accessed on 15 March 2020).
11. 1.4. Support Vector Machines—Scikit-Learn 0.22.2 Documentation. Available online: https://scikit-learn.org/stable/modules/svm.html (accessed on 5 March 2020).
12. Guestrin, C. SVMs, Duality and the Kernel Trick. *Mach. Learn.* **2006**, *10701*, 15781.
13. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks* **2015**, *61*, 85–117.
14. Lecun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444

15. Neapolitan, R.E.; Jiang, X. Neural Networks and Deep Learning. In *Artificial Intelligence*; CRC Press Taylor& Francis Group: Boca Raton, FL, USA, 2018.
16. Jordan, M.I.; Bishop, C.M. Neural networks. In *Computer Science Handbook*, 2nd ed.; CRC Press: Boca Raton, FL, USA, 2004; ISBN 9780203494455.
17. Merenda, M.; Praticò, F.G.; Fedele, R.; Carotenuto, R.; Corte, D.; Della Corte, F.G. A Real-Time Decision Platform for the Management of Structures and Infrastructures. *Electronics* **2019**, *8*, 1180.
18. Anandhalli, M.; Baligar, V.P. A novel approach in real-time vehicle detection and tracking using Raspberry Pi. *Alex. Eng. J.* **2018**, *57*, 1597–1607.
19. Arasteh, H.; Hosseinnezhad, V.; Loia, V.; Tommasetti, A.; Troisi, O.; Shafie-Khah, M.; Siano, P. Iot-based smart cities: A survey. In Proceedings of the 2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC), Florence, Italy, 7–10 June 2016; pp. 2–7.
20. Zanella, A.; Bui, N.; Castellani, A.; Vangelista, L.; Zorzi, M. Internet of Things for Smart Cities. *IEEE Internet Things J.* **2014**, *1*, 22–32.
21. Bibri, S.E. The IoT for smart sustainable cities of the future: An analytical framework for sensor-based big data applications for environmental sustainability. *Sustain. Cities Soc.* **2018**, *38*, 230–253.
22. Kim, T. hoon; Ramos, C.; Mohammed, S. Smart City and IoT. *Futur. Gener. Comput. Syst.* **2017**, *76*, 159–162.
23. Sajjad, M.; Nasir, M.; Muhammad, K.; Khan, S.; Jan, Z.; Sangaiah, A.K.; Elhoseny, M.; Baik, S.W. Raspberry Pi assisted face recognition framework for enhanced law-enforcement services in smart cities. *Futur. Gener. Comput. Syst.* **2017**, *108*, 995–1007.
24. Zhang, T.; Chowdhery, A.; Bahl, P.; Jamieson, K.; Banerjee, S. The design and implementation of a wireless video surveillance system. In Proceedings of the Annual International Conference on Mobile Computing and Networking, MobiCom'15: The 21th Annual International Conference on Mobile Computing and Networking, Paris, France, 7–11 September 2015.
25. Borgia, E. The Internet of Things vision: Key features, applications and open issues. *Comput. Commun.* **2014**, *54*, 1–31.
26. Magrini, M.; Moroni, D.; Palazzese, G.; Pieri, G.; Leone, G.; Salvetti, O. Computer Vision on Embedded Sensors for Traffic Flow Monitoring. In Proceedings of the 2015 IEEE 18th International Conference on Intelligent Transportation Systems, Las Palmas, Spain, 15–18 September 2015; pp. 161–166.
27. Arshad, B.; Ogie, R.; Barthelemy, J.; Pradhan, B.; Verstaevel, N.; Perez, P. Computer Vision and IoT-Based Sensors in Flood Monitoring and Mapping: A Systematic Review. *Sensors* **2019**, *19*, 5012.
28. Fafoutis, X.; Marchegiani, L.; Elsts, A.; Pope, J.; Piechocki, R.; Craddock, I. Extending the battery lifetime of wearable sensors with embedded machine learning. In Proceedings of the 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), Singapore, Singapore, 5–8 February 2018; pp. 269–274.
29. Ieracitano, C.; Mammone, N.; Hussain, A.; Morabito, F.C. A novel multi-modal machine learning based approach for automatic classification of EEG recordings in dementia. *Neural Networks* **2020**, *123*, 176–190.
30. Rajkomar, A.; Dean, J.; Kohane, I. Machine learning in medicine. *N. Engl. J. Med.* **2019**, *380*, 1347–1358.
31. Ngiam, K.Y.; Khor, I.W. Big data and machine learning algorithms for health-care delivery. *Lancet Oncol.* **2019**, *20*, e262–e273.
32. Beam, A.L.; Kohane, I. Big Data and Machine Learning in Health Care. *JAMA* **2018**, *319*, 1317.
33. Amato, F.; López, A.; Peña-Méndez, E.M.; Vanhara, P.; Hampl, A.; Havel, J. Artificial neural networks in medical diagnosis. *J. Appl. Biomed.* **2013**, *11*, 47–58.
34. Syafrudin, M.; Alfian, G.; Fitriyani, N.L.; Rhee, J. Performance Analysis of IoT-Based Sensor, Big Data Processing, and Machine Learning Model for Real-Time Monitoring System in Automotive Manufacturing. *Sensors* **2018**, *18*, 2946.
35. Kihei, B.; Copeland, J.A.; Chang, Y. Automotive Doppler sensing: The Doppler profile with machine learning in vehicle-to-vehicle networks for road safety. In Proceedings of the IEEE Workshop on Signal Processing Advances in Wireless Communications, SPAWC, Sapporo, Japan, 3–6 July 2017.
36. Gharib, M.; Lollini, P.; Botta, M.; Amparore, E.; Donatelli, S.; Bondavalli, A. On the Safety of Automotive Systems Incorporating Machine Learning Based Components: A Position Paper. In Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN-W 2018, Luxembourg City, Luxembourg, 25–28 June 2018.
37. Luckow, A.; Kennedy, K.; Manhardt, F.; Djerekarov, E.; Vorster, B.; Apon, A. Automotive big data: Applications, workloads and infrastructures. In Proceedings of the 2015 IEEE International Conference on Big Data, IEEE Big Data 2015, Santa Clara, CA, USA, 29 October–1 November 2015.

38. OpenCV. Available online: https://opencv.org/ (accessed on 3 January 2020).
39. Viola, P; Jones, M.J. *Robust Real-Time Object Detection*; Technical Reports; Cambridge Research Laboratory: Cambridge, MA, USA, 2001.
40. Rublee, E.; Rabaud, V.; Konolige, K.; Bradski, G. ORB: An efficient alternative to SIFT or SURF. In Proceedings of the IEEE International Conference on Computer Vision, Barcelona, Spain, 6–13 November 2011.
41. Shashua, A. Introduction to Machine Learning: Class Notes 67577. *arXiv* **2009**, arXiv:0904.3664.
42. Transactions, E.A.I.E.; Health, P. Designing wearable sensing platforms for healthcare in a residential environment. *EAI Endorsed Trans. Pervasive Health Technol.* **2017**, *3*, 12.
43. A. Shoeb, D. Carlson, E. Panken and T. Denison, A micropower support vector machine based seizure detection architecture for embedded medical devices. In Proceedings of the 2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Minneapolis, MN, USA, 3–6 September 2009; pp. 4202-4205.
44. Lee, D.D.; Seung, H.S. Learning in intelligent embedded systems. In *WOES'99: Proceedings of the Workshop on Embedded Systems on Workshop on Embedded Systems*; USENIX Association: Berkeley, CA, USA, 1999; p. 9.
45. Li, H.; Ota, K.; Dong, M. Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing. *IEEE Netw.* **2018**, *32*, 96–101.
46. Yazici, M.; Basurra, S.; Gaber, M. Edge Machine Learning: Enabling Smart Internet of Things Applications. *Big Data Cogn. Comput.* **2018**, *2*, 26.
47. Praticò, F.G.; Della Corte, F.G.; Merenda, M. Self-powered sensors for road pavements. In Proceedings of the Functional Pavement Design—Proceedings of the 4th Chinese-European Workshop on Functional Pavement Design, CEW 2016, Delft, The Netherlands, 29 June–1 July 2016.
48. Iero, D.; Della Corte, F.G.; Felini, C.; Merenda, M.; Minarini, C.; Rubino, A. RF-Powered UHF-RFID Analog Sensors Platform. In Proceedings of the 2015 XVIII AISEM Annual Conference, Trento, Italy, 3–5 February 2015.
49. Fedele, R.; Merenda, M.; Giammaria, F.; Praticò, F.G. Energy harvesting for IoT road monitoring systems. *Instrumentation Mesure Métrologie* **2018**, *18*, 17.
50. Merenda, M.; Iero, D.; Pangallo, G.; Falduto, P.; Adinolfi, G.; Merola, A.; Graditi, G.; Della Corte, F.G. Open-Source Hardware Platforms for Smart Converters with Cloud Connectivity. *Electron.* **2019**, *8*, 367.
51. Della Corte, F.G.; Merenda, M.; Bellizzi, G.G.; Isernia, T.; Carotenuto, R. Temperature Effects on the Efficiency of Dickson Charge Pumps for Radio Frequency Energy Harvesting. *IEEE Access* **2018**, *6*, 65729–65736.
52. Tatarinova, T.V.; Editors, Y.N.; Raschka, S.; Verdier, C.F.J.E.S.O.; Hearty, J.; Huffman, J.; Pajankar, A. *Python Machine Learning*. Packt Publishing: Birmingham, UK, September 2015, ISBN:978-1-78439-390-8
53. Kingma, D.P.; Ba, J.L. Adam: A method for stochastic optimization. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015—Conference Track Proceedings, San Diego, CA, USA, 7–9 May 2015.
54. Krizhevsky, A.; Sutskever, I.; Hinton, G.E.; Pdf ImageNet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90.
55. Caterini, A.L.; Chang, D.E. Recurrent neural networks. In *SpringerBriefs in Computer Science*; Springer International Publishing: New York, NY, USA, 2018; doi:10.1007/978-3-319-70338-1
56. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.* **2015**, *115*, 211–252.
57. Schuster, M.; Paliwal, K.K. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.* **1997**, *45*, 2673–2681.
58. Beil, J.; Perner, G.; Asfour, T. Speech Recognition With Deep Recurrent Neural Networks. In Proceedings of the IEEE International Conference on Rehabilitation Robotics, Singapore, 1–14 August 2015; pp. 119–124.
59. Hochreiter, S.; Urgen Schmidhuber, J. Long Shortterm Memory. *Neural Comput.* **1997**, *9*, 1735–1780.
60. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Genrative Adversial Nets., In Proceedings of the 27th International Conference on Neural Information Processing Systems–Volume 2, Montreal, QC, Canada, 8–13 December 2014; pp.2672–2680.

61. Altman, N.S. An introduction to kernel and nearest-neighbor nonparametric regression. *Am. Stat.* **1992**, *46*, 175–185.
62. Gupta, C.; Suggala, A.S.; Goyal, A.; Simhadri, H.V.; Paranjape, B.; Kumar, A.; Goya, S.; Udupa, R.; Varma, M.; Jain, P. ProtoNN: Compressed and accurate kNN for resource-scarce devices. In Proceedings of the 34th International Conference on Machine Learning ICML, Sydney, Australia, 6–11 August 2017; Volume 3, pp. 2144–2159.
63. Gope, D.; Dasika, G.; Mattina, M. Ternary Hybrid Neural-Tree Networks for Highly Constrained IoT Applications. *arXiv* **2019**, arXiv:1903.01531.
64. Kumar, A.; Goyal, S.; Varma, M. Resource-efficient machine learning in 2 KB RAM for the Internet of Things. In Proceedings of the 34th International Conference on Machine Learning ICML, Sydney, Australia, 6–11 August 2017; Volume 4, pp. 3062–3071.
65. Haigh, K.Z.; Mackay, A.M.; Cook, M.R.; Lin, L.G. *Machine Learning for Embedded Systems : A Case Study*; Technical Report; BBN Technologies: Cambridge, MA, USA, 2015.
66. Chen, J.; Ran, X. Deep Learning With Edge Computing: A Review. *Proc. IEEE* **2019**, *107*, 1655–1674.
67. Sze, V.; Chen, Y.H.; Emer, J.; Suleiman, A.; Zhang, Z. Hardware for machine learning: Challenges and opportunities. In Proceedings of the 2017 IEEE Custom Integrated Circuits Conference (CICC), Austin, TX, USA, 30 April–3 May 2017; pp.1–8.
68. 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* **2017**, arXiv:1704.04861.
69. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv* **2016**, arXiv:1602.07360; pp. 1–13.
70. Han, S.; Mao, H.; Dally, W.J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv* **2015**, arXiv:1510.00149; pp. 1–14.
71. TensorFlow. Available online: https://www.tensorflow.org/ (accessed on 11 November 2019).
72. Home—Keras Documentation. Available online: https://keras.io/ (accessed on 11 November 2019).
73. Yao, S.; Zhao, Y.; Zhang, A.; Su, L.; Abdelzaher, T. DeepIoT: Compressing Deep Neural Network Structures for Sensing Systems with a Compressor-Critic Framework. In Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems, Delft, The Netherlands, 6–8 November 2017.
74. Molchanov, P.; Tyree, S.; Karras, T.; Aila, T.; Kautz, J. Pruning Convolutional Neural Networks for Resource Efficient Inference. *arXiv* **2016**, arXiv:1611.06440; pp. 6–10.
75. Anwar, S.; Sung, W. Compact Deep Convolutional Neural Networks With Coarse Pruning. **2016**, *1*, 1–10.
76. Yang, T.J.; Chen, Y.H.; Sze, V. Designing energy-efficient convolutional neural networks using energy-aware pruning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 6071–6079.
77. Narang, S.; Elsen, E.; Diamos, G.; Sengupta, S. Exploring Sparsity in Recurrent Neural Networks. *arXiv* **2017**, arXiv:1704.05119; pp. 1–10.
78. Guo, Y.; Yao, A.; Chen, Y. Dynamic network surgery for efficient DNNs. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*; Curran Associates Inc.: Red Hook, NY, USA, 2016; pp. 1387–1395.
79. Hinton, G.; Vinyals, O.; Dean, J. Distilling the Knowledge in a Neural Network. *arXiv* **2015**, arXiv:1503.02531; pp. 1–9.
80. X-CUBE-AI—AI Expansion Pack for STM32CubeMX—STMicroelectronics. Available online: https://www.st.com/en/embedded-software/x-cube-ai.html#overview (accessed on 18 November 2019).
81. TensorFlow Lite for Microcontrollers. Available online: https://www.tensorflow.org/lite/microcontrollers (accessed on 15 March 2020).
82. Arduino Nano 33 BLE Sense with Headers | Arduino Official Store. Available online: https://store.arduino.cc/arduino-nano-33-ble-sense-with-headers (accessed on 15 March 2020).
83. SparkFun Edge Development Board—Apollo3 Blue—DEV-15170—SparkFun Electronics. Available online: https://www.sparkfun.com/products/15170 (accessed on 15 March 2020).
84. Artificial Intelligence (AI)—STMicroelectronics. Available online: https://www.st.com/content/st_com/en/about/innovation---technology/artificial-intelligence.html (accessed on 15 March 2020).

85. Adafruit EdgeBadge—TensorFlow Lite for Microcontrollers ID: 4400—$35.95 : Adafruit Industries, Unique & fun DIY electronics and kits. Available online: https://www.adafruit.com/product/4400 (accessed on 15 March 2020).
86. Overview | Espressif Systems. Available online: https://www.espressif.com/en/products/hardware/esp32-devkitc/overview (accessed on 15 March 2020).
87. Overview | Espressif Systems. Available online: https://www.espressif.com/en/products/hardware/esp-eye/overview (accessed on 15 March 2020).
88. High-Performing AI Solutions to Transform our Digital World—Arm. Available online: https://www.arm.com/solutions/artificial-intelligence (accessed on 15 March 2020).
89. New AI technology from Arm delivers intelligence for IoT—Arm. Available online: https://www.arm.com/company/news/2020/02/new-ai-technology-from-arm (accessed on 15 March 2020).
90. Zhao, Z.; Barijough, K.M.; Gerstlauer, A. DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **2018**, *37*, 2348–2359.
91. Nikouei, S.Y.; Chen, Y.; Song, S.; Xu, R.; Choi, B.Y.; Faughnan, T. Smart surveillance as an edge network service: From harr-cascade, SVM to a Lightweight CNN. In Proceedings of the 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC), Philadelphia, PA, USA, 18–20 October 2018; pp. 256–265.
92. Xu, R.; Nikouei, S.Y.; Chen, Y.; Polunchenko, A.; Song, S.; Deng, C.; Faughnan, T.R. Real-Time Human Objects Tracking for Smart Surveillance at the Edge. In Proceedings of the IEEE International Conference on Communications, Kansas City, MO, USA, 20–24 May 2018..
93. Chand, G.; Ali, M.; Barmada, B.; Liesaputra, V.; Ramirez-Prado, G. Tracking a person's behaviour in a smart house. In *International Conference on Service-Oriented Computing*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2019.
94. Rosato, D.; Comai, S.; Masciadri, A.; Salice, F. Non-invasive monitoring system to detect sitting people. In Proceedings of the 4th EAI International Conference on Smart Objects and Technologies for Social Good, Bologna, Italy, 28–30 November 2018; pp. 261–264.
95. SparkFun Edge Hookup Guide—learn.sparkfun.com. Available online: https://learn.sparkfun.com/tutorials/sparkfun-edge-hookup-guide/all (accessed on 14 April 2020).
96. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge Computing: Vision and Challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646.
97. Yu-han, T.; Ravindranath, L.; Deng, S.; Chen, T.Y. Continuous, Real-Time Object Recognition on Mobile Devices Categories and Subject Descriptors, In Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys '15). Association for Computing Machinery, New York, NY, USA, 2015; pp. 155–168.
98. Hung, C.C.; Ananthanarayanan, G.; Bodik, P.; Golubchik, L.; Yu, M.; Bahl, P.; Philipose, M. VideoEdge: Processing camera streams using hierarchical clusters. In Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing (SEC), Seattle, WA, USA, 25–27 October 2018; pp. 115–131.
99. Jiang, A.H.; Wong, D.L.K.; Canel, C.; Tang, L.; Misra, I.; Kaminsky, M.; Kozuch, M.A.; Pillai, P.; Labs, I.; Andersen, D.G.; et al. Mainstream : Dynamic Stem-Sharing for Multi-Tenant Video Processing. In Proceedings of the 2018 USENIX Annual Technical Conference (USENIX ATC 18), Boston, MA, USA, 11–13 July 2018.
100. Radio Regulations. Available online: https://www.itu.int/pub/R-REG-RR/en (accessed on 22 February 2020).
101. Radio Versions | Bluetooth® Technology Website. Available online: https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/radio-versions/ (accessed on 11 March 2020).
102. Dekimpe, R.; Xu, P.; Schramme, M.; Flandre, D.; Bol, D. A Battery-Less BLE IoT Motion Detector Supplied by 2.45-GHz Wireless Power Transfer. In Proceedings of the 2018 IEEE 28th International Symposium on Power and Timing Modeling, Optimization and Simulation, PATMOS, Platja d'Aro, Spain, 2–4 July 2018; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2018; pp. 68–75.
103. Bluetooth 5: Go Faster, Go Further. Available online: https://www.bluetooth.com/wp-content/uploads/2019/03/Bluetooth_5-FINAL.pdf (accessed on 9 April 2020).
104. Bluetooth Special Interest Group (SIG). Bluetooth Core Specification Version 5.0. In *Bluetooth Core Specif. Version 5.2*; Bluetooth Special Interest Group (SIG): Kirkland, WA, USA, 2019. Available online:

https://www.bluetooth.com/wp-content/uploads/2020/01/Bluetooth_5.2_Feature_Overview.pdf (accessed on 20 April 2020).
105. Zigbee Alliance Website. Available online: https://zigbeealliance.org/ (accessed on 11 March 2020).
106. 802.15.4v-2017—IEEE Standard for Low-Rate Wireless Networks—Amendment 5: Enabling/Updating the Use of Regional Sub-GHz Bands. Available online: https://standards.ieee.org/standard/802_15_4v-2017.html (accessed on 11 March 2020).
107. Pan, M.S.; Tseng, Y.C. ZigBee and Their Applications. In *Sensor Networks and Configuration, Fundamentals, Standards, Platforms, and Applications*; Springer: Berlin/Heidelberg, Germany, 2007; Volume 16, pp. 349–368.
108. Islam, M.L.; Faizan, C.; Quavi, S.M.A. IOT Based Smart Garbage Monitoring System. *Int. J. Comput. Sci. Eng.* **2019**, *7*, 649–651.
109. Yaqoob, I.; Hashem, I.A.T.; Mehmood, Y.; Gani, A.; Mokhtar, S.; Guizani, S. Enabling communication technologies for smart cities. *IEEE Commun. Mag.* **2017**, *55*, 112–120.
110. Sahitya, G.; Balaji, N.; Naidu, C.D.; Abinaya, S. Designing a wireless sensor network for precision agriculture using zigbee. In Proceedings of the 7th IEEE International Advanced Computing Conference, IACC 2017, Hyderabad, India, 5–7 January 2017.
111. Hidayat, T. Internet of Things Smart Agriculture on ZigBee: A Systematic Review. *J. Telekomun. dan Komp'ut.* **2017**, *8*, 75–86.
112. Lei, Y.; Wang, T.; Wu, J. Vehicles relative positioning based on ZigBee and GPS technology. In Proceedings of the ICEIEC 2016 IEEE 6th International Conference on Electronics Information and Emergency Communication, Beijing, China, 17–19 June 2016.
113. Dong, C.; Chen, X.; Dong, H.; Yang, K.; Guo, J.; Bai, Y. Research on intelligent vehicle infrastructure cooperative system based on zigbee. In Proceedings of the 2019 5th International Conference on Transportation Information and Safety (ICTIS), Liverpool, UK, 14–17 July 2019; pp. 1337–1343.
114. Lee, H.J.; Lee, S.H.; Ha, K.S.; Jang, H.C.; Chung, W.-Y.C.; Kim, J.Y.; Chang, Y.-S.; Yoo, D.H. Ubiquitous healthcare service using Zigbee and mobile phone for elderly patients. *Int. J. Med Inform.* **2009**, *78*, 193–198.
115. Chae, M.J.; Yoo, H.; Kim, J.; Cho, M. Development of a wireless sensor network system for suspension bridge health monitoring. *Autom. Constr.* **2012**, *21*, 237–252.
116. Z-Wave | Safer, Smarter Homes Start with Z-Wave. Available online: https://www.z-wave.com/ (accessed on 12 March 2020).
117. ANT Protocol | Dynastream Innovations. Available online: https://www.dynastream.com/solutions/ant-wireless/ (accessed on 9 April 2020).
118. What is ANT+—THIS IS ANT. Available online: https://www.thisisant.com/consumer/ant-101/what-is-ant/ (accessed on 16 April 2020).
119. Mulligan, G. The 6LoWPAN architecture. In Proceedings of the 4th Workshop on Embedded Networked Sensors, EmNets, Cork, Ireland, 25–26 June 2007.
120. Unwala, I.; Taqvi, Z.; Lu, J. Thread: An IoT protocol. In Proceedings of the IEEE Green Technologies Conference, Austin, TX, USA, 4–6 April 2018.
121. Shop Humidor Monitoring from Smartphone and Tablet Habueno. Available online: https://www.habueno.com/shop/?lang=en (accessed on 16 April 2020).
122. WiFi HaLow | WiFi Alliance. Available online: https://www.WiFi.org/discover-WiFi/WiFi-halow (accessed on 12 March 2020).
123. Flores, A.B.; Guerra, R.E.; Knightly, E.W.; Ecclesine, P.; Pandey, S. IEEE 802.11af: A standard for TV white space spectrum sharing. *IEEE Commun. Mag.* **2013**, *51*, 92–100.
124. Bellalta, B. IEEE 802.11ax: High-efficiency WLANS. *IEEE Wirel. Commun.* **2016**, *23*, 38–46.
125. Merenda, M.; Iero, D.; Della Corte, F.G. CMOS RF Transmitters with On-Chip Antenna for Passive RFID and IoT Nodes. *Electron.* **2019**, *8*, 1448.
126. Lazaro, A.; Villarino, R.; Girbau, D. A Survey of NFC Sensors Based on Energy Harvesting for IoT Applications. *Sensors* **2018**, *18*, 3746.
127. LoRa Alliance® Website. Available online: https://lora-alliance.org/ (accessed on 11 March 2020).
128. Chiani, M.; Elzanaty, A. On the LoRa Modulation for IoT: Waveform Properties and Spectral Analysis. *IEEE Internet Things J.* **2019**, *6*, 8463–8470.
129. Augustin, A.; Yi, J.; Clausen, T.H.; Townsley, W.M. A Study of LoRa: Long Range & Low Power Networks for the Internet of Things. *Sensors* **2016**, *16*, 1466.

130. Suresh, V.M.; Sidhu, R.; Karkare, P.; Patil, A.; Lei, Z.; Basu, A. Powering the IoT through embedded machine learning and LoRa. In Proceedings of the IEEE World Forum on Internet of Things, WF-IoT, Singapore, 5–8 February 2018.
131. Merenda, M.; Felini, C.; Della Corte, F.G. A Monolithic Multisensor Microchip with Complete On-Chip RF Front-End. *Sensors* **2018**, *18*, 110.
132. Sigfox—The Global Communications Service Provider for the Internet of Things (IoT). Available online: https://www.sigfox.com/en (accessed on 5 January 2020).
133. Huang, J.; Qian, F.; Guo, Y.; Zhou, Y.; Xu, Q.; Mao, Z.M.; Sen, S.; Spatscheck, O. An in-depth study of LTE: Effect of network protocol and application behavior on performance. *Comput. Commun. Rev.* **2013**, *43*, 363–374.
134. Akpakwu, G.A.; Silva, B.J.; Hancke, G.P.; Abu-Mahfouz, A.M. A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges. *IEEE Access* **2018**, *6*, 3619–3647.
135. Li, S.; Xu, L. Da; Zhao, S. 5G Internet of Things: A survey. *J. Ind. Inf. Integr.* **2018**, *10*, 1–9.
136. GSMA | Narrowband – Internet of Things (NB-IoT) | Internet of Things. Available online: https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot/ (accessed on 11 March 2020).
137. Ratasuk, R.; Mangalvedhe, N.; Zhang, Y.; Robert, M.; Koskinen, J.P. Overview of narrowband IoT in LTE Rel-13. In Proceedings of the 2016 IEEE Conference on Standards for Communications and Networking (CSCN), Berlin, Germany, 31 October–2 November 2016.
138. Borkar, S.R. Long-term evolution for machines (LTE-M). *LPWAN Technol. IoT M2M Appl.* **2020**, 145–166.
139. Wang, D.; Chen, D.; Song, B.; Guizani, N.; Yu, X.; Du, X. From IoT to 5G I-IoT: The Next Generation IoT-Based Intelligent Algorithms and 5G Technologies. *IEEE Commun. Mag.* **2018**, *56*, 114–120.
140. Morocho-Cayamcela, M.E.; Lee, H.; Lim, W. Machine learning for 5G/B5G mobile and wireless communications: Potential, limitations, and future directions. *IEEE Access* **2019**, *7*, 137184–137206.
141. Al-Sarawi, S.; Anbar, M.; Alieyan, K.; Alzubaidi, M. Internet of Things (IoT) communication protocols: Review. In Proceedings of the ICIT 2017—8th International Conference on Information Technology, Amman, Jordan, 17–18 May 2017.
142. Mahmoud, M.S.; Mohamad, A.A.H. A Study of Efficient Power Consumption Wireless Communication Techniques/ Modules for Internet of Things (IoT) Applications. *Adv. Internet Things* **2016**.
143. Mekki, K.; Bajic, E.; Chaxel, F.; Meyer, F. A comparative study of LPWAN technologies for large-scale IoT deployment. *ICT Express* **2019**, *5*, 1–7.
144. Choi, C.S.; Jeong, J.D.; Lee, I.W.; Park, W.K. LoRa based renewable energy monitoring system with open IoT platform. In Proceedings of the International Conference on Electronics, Information and Communication, ICEIC, Honolulu, HI, USA, 24–27 January 2018.
145. Zhou, Q.; Zheng, K.; Hou, L.; Xing, J.; Xu, R. Design and Implementation of Open LoRa for IoT. *IEEE Access* **2019**, *7*, 100649–100657.
146. Wang, S.Y.; Chen, Y.R.; Chen, T.Y.; Chang, C.H.; Cheng, Y.H.; Hsu, C.C.; Lin, Y.B. Performance of LoRa-based IoT applications on campus. In Proceedings of the IEEE Vehicular Technology Conference, Toronto, ON, Canada, 24–27 September 2017.
147. Sarker, V.K.; Queralta, J.P.; Gia, T.N.; Tenhunen, H.; Westerlund, T. A survey on LoRa for IoT: Integrating edge computing. In Proceedings of the 2019 4th International Conference on Fog and Mobile Edge Computing, FMEC, Rome, Italy, 10–13 June 2019.
148. Poursafar, N.; Alahi, M.E.E.; Mukhopadhyay, S. Long-range wireless technologies for IoT applications: A review. In Proceedings of the International Conference on Sensing Technology, ICST, Sydney, Australia, 4–6 December 2017.
149. Vejlgaard, B.; Lauridsen, M.; Nguyen, H.; Kovacs, I.Z.; Mogensen, P.; Sorensen, M. Coverage and Capacity Analysis of Sigfox, LoRa, GPRS, and NB-IoT. In Proceedings of the IEEE Vehicular Technology Conference, Sydney, Australia, 4–7 June 2017.
150. Ray, B. NB-IoT vs. LoRa vs. Sigfox, Available online: https://www.link-labs.com/blog/nb-iot-vs-lora-vs-sigfox (accessed on 16 April 2020).
151. Zuniga, J.C.; Ponsard, B. Sigfox System Description. *Ietf 97*, Available online: https://datatracker.ietf.org/meeting/97/materials/slides-97-lpwan-25-sigfox-system-description-00 (accessed on 16 April 2020).

152. Froiz-Míguez, I.; Fernandez-Carames, T.M.; Fraga-Lamas, P.; Castedo, L. Design, Implementation and Practical Evaluation of an IoT Home Automation System for Fog Computing Applications Based on MQTT and ZigBee-WiFi Sensor Nodes. *Sensors* **2018**, *18*, 2660.
153. Ergen, S.C. ZigBee/IEEE 802.15.4 Summary. UC Berkeley September 2004. Available Online: http://users.eecs.northwestern.edu/~peters/references/ZigtbeeIEEE802.pdf (accessed on 16 April 2020).
154. Li, Y.; Chi, Z.; Liu, X.; Zhu, T. Passive-ZigBee: Enabling zigbee communication in IoT networks with 1000X+ less power consumption. In Proceedings of the SenSys 2018—Proceedings of the 16th Conference on Embedded Networked Sensor Systems, Shenzhen, China, 4–7 November 2018.
155. Patil, S.M.; Moiz Baig, M., Survey on Creating ZigBee Chain Reaction Using IoT. *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.* **2018**, *3*, 545–549.
156. Hersent, O.; Boswarthick, D.; Elloumi, O. Z-Wave. In *The Internet of Things: Key Applications and Protocols*; John Wiley & Sons: Hoboken, NJ, USA, 2011; ISBN 9781119994350.
157. Raza, S.; Misra, P.; He, Z.; Voigt, T. Building the Internet of Things with bluetooth smart. *Ad Hoc Networks* **2017**, *57*, 19–31.
158. Cha, S.-C.; Chen, J.-F.; Su, C.; Yeh, K.-H. A Blockchain Connected Gateway for BLE-Based Devices in the Internet of Things. *IEEE Access* **2018**, *6*, 24639–24649.
159. Jeon, K.E.; She, J.; Soonsawad, P.; Ng, P.C. BLE Beacons for Internet of Things Applications: Survey, Challenges, and Opportunities. *IEEE Internet Things J.* **2018**, *5*, 811–828.
160. Collotta, M.; Pau, G.; Talty, T.; Tonguz, O.K. Bluetooth 5: A Concrete Step Forward toward the IoT. *IEEE Commun. Mag.* **2018**, *56*, 125–131.
161. Ray, P.P.; Agarwal, S. Bluetooth 5 and Internet of Things: Potential and architecture. In Proceedings of the International Conference on Signal Processing, Communication, Power and Embedded System, SCOPES 2016, Paralakhemundi, India, 3–5 October 2016.
162. Pau, G.; Collotta, M.; Maniscalco, V. Bluetooth 5 Energy Management through a Fuzzy-PSO Solution for Mobile Devices of Internet of Things. *Energies* **2017**, *10*, 992.
163. López-Matencio, P.; Vales-Alonso, J.; Costa-Montenegro, E. ANT: Agent Stigmergy-Based IoT-Network for Enhanced Tourist Mobility. *Mob. Inf. Syst.* **2017**, *2017*, 1–15.
164. Shrivastava, V.; Rayanchu, S.; Yoon, J.; Banerjee, S. 802.11n under the microscope. In Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC, Vouliagmeni, Greece, 20–22 October 2008.
165. IEEE. *IEEE Std 802.11ah-2016 (Amendment to IEEE Std 802.11-2016, as Amended by IEEE Std 802.11ai-2016). IEEE Standard for Information Technology--Telecommunications and Information Exchange between Systems—Local and metropolitan Area Networks--Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 2: Sub 1 GHz License Exempt Operation*; IEEE Computer Society: Washington, DC, USA, 14 December 2016; doi: 10.1109/IEEESTD.2016.7786995
166. Park, M. IEEE 802.11ah: sub-1-GHz license-exempt operation for the internet of things. *IEEE Commun. Mag.* **2015**, *53*, 145–151.
167. Hossain, M.I.; Lin, L.; Markendahl, J. A Comparative Study of IoT-Communication Systems Cost Structure: Initial Findings of Radio Access Networks Cost. In Proceedings of the 11th CMI International Conference, 2018: Prospects and Challenges Towards Developing a Digital Economy within the EU, PCTDDE 2018, Copenhagen, Denmark, 29–30 November 2018.
168. Chen, M.; Miao, Y.; Hao, Y.; Hwang, K. Narrow Band Internet of Things. *IEEE Access* **2017**, *5*, 20557–20577.
169. Sara, J.J.; Hossain, M.S.; Khan, W.Z.; Aalsalem, M.Y. Survey on Internet of Things and 4G. In Proceedings of the 2019 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET), Tangerang, Indonesia, 23–24 October 2019; pp. 1–6.
170. Gemalto Introducing 5G networks—Characteristics and Usages. 2016. Available online: Https://www.Gemalto.Com (accessed on 16 April 2020).
171. Martinez, I.S.H.; Salcedo, I.P.O.J.; Daza, I.B.S.R. IoT application of WSN on 5G infrastructure. In Proceedings of the 2017 International Symposium on Networks, Computers and Communications, ISNCC, Marrakech, Morocco, 16–18 May 2017.
172. Mumtaz, S.; Bo, A.; Al-Dulaimi, A.; Tsang, K.F. Guest Editorial 5G and beyond Mobile Technologies and Applications for Industrial IoT (IIoT). *IEEE Trans. Ind. Inf.* **2018**, *14*,2588–2591.
173. Kang, Y.; Hauswald, J.; Gao, C.; Rovinski, A.; Mudge, T.; Mars, J.; Tang, L. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In Proceedings of the Twenty-Second International

Conference on Architectural Support for Programming Languages and Operating Systems, Xi'an China, 8–12 April 2017; pp. 615–629,

174. Teerapittayanon, S.; McDanel, B.; Kung, H.T. Distributed Deep Neural Networks over the Cloud, the Edge and End Devices. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 328–339.
175. Wang, J.; Zhu, X.; Zhang, J.; Cao, B.; Bao, W.; Yu, P.S. Not just privacy: Improving performance of private deep learning in mobile cloud. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, UK, 19–23 August 2018; pp. 2407–2416.
176. Du, M.; Wang, K.; Xia, Z.; Zhang, Y. Differential Privacy Preserving of Training Model in Wireless Big Data with Edge Computing. *IEEE Trans. Big Data* **2018**, doi:10.1109/TBDATA.2018.2829886.
177. Abadi, M.; McMahan, H.B.; Chu, A.; Mironov, I.; Zhang, L.; Goodfellow, I.; Talwar, K. Deep learning with differential privacy. In Proceedings of the ACM Conference on Computer and Communications Security, Vienna, Austria, 25–27 October 2016.
178. Xu, C.; Ren, J.; Zhang, D.; Zhang, Y. Distilling at the Edge: A Local Differential Privacy Obfuscation Framework for IoT Data Analytics. *IEEE Commun. Mag.* **2018**, *56*, 20–25.
179. Miao, Q.; Jing, W.; Song, H. Differential privacy–based location privacy enhancing in edge computing. *Concurr. Comput. Pr. Exp.* **2018**, *31*, e4735.
180. Dowlin, N.; Edu, N.; Gilad-Bachrach, R.; Laine, K.; Lauter, K.; Naehrig, M.; Wernsing, J.; Com, J.W. CryptoNets: Applying neural networks to Encrypted data with high throughput and accuracy—Microsoft research. *Microsoft Res. TechReport* **2016**, *48*, 1–12.
181. Dias, M.; Abad, A.; Trancoso, I. Exploring Hashing and Cryptonet Based Approaches for Privacy-Preserving Speech Emotion Recognition. In Proceedings of the ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing, Calgary, AB, Canada, 15–20 April 2018.
182. Morris, A.; Mellis, C. Privacy-preserving classifification on deep neural network. *IACR Cryptol. ePrint Arch.* **2017**, 2017, 35.
183. Mao, J.; Chen, X.; Nixon, K.W.; Krieger, C.; Chen, Y. MoDNN: Local distributed mobile computing system for Deep Neural Network. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, Lausanne, Switzerland, 27–31 March 2017; pp. 1396–1401.
184. Chandakkar, P.S.; Li, Y.; Ding, P.L.K.; Li, B. Strategies for Re-Training a Pruned Neural Network in an Edge Computing Paradigm. In Proceedings of the 2017 IEEE International Conference on Edge Computing (EDGE), Honolulu, HI, USA, 25–30 June 2017; pp. 244–247.
185. Tao, Z.; Li, Q. eSGD: Communication efficient distributed deep learning on the edge. In Proceedings of the USENIX Workshop on Hot Topics in Edge Computing, HotEdge 2018, Co-Located with USENIX ATC 2018, Boston, MA, USA, 10 July 2018.
186. Deep Learning Hardware: FPGA vs. GPU. Available online: https://semiengineering.com/deep-learning-hardware-fpga-vs-gpu/ (accessed on 18 February 2020).
187. cuBLAS | NVIDIA Developer. Available online: https://developer.nvidia.com/cublas (accessed on 21 February 2020).
188. NVIDIA cuDNN | NVIDIA Developer. Available online: https://developer.nvidia.com/cudnn (accessed on 21 February 2020).
189. Mathieu, M.; Henaff, M.; LeCun, Y. Fast Training of Convolutional Networks through FFTs. *arXiv* **2013**, arXiv:1312.5851; pp. 1–9.
190. Cong, J.; Xiao, B. Minimizing in Convolutional Neural Networks. *Int. Conf. Artif. Neural Networks* **2014**, *8681*, 281–290.
191. Lavin, A.; Gray, S. Fast Algorithms for Convolutional Neural Networks. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016.
192. Hailo–Top Performing AI Chip for Edge Devices. Available online: https://hailo.ai/ (accessed on 18 November 2019).
193. Cloud TPU | Google Cloud. Available online: https://cloud.google.com/tpu/ (accessed on 18 November 2019).
194. US8655815B2—Neural processing unit—Google Patents. Available online: https://patents.google.com/patent/US8655815B2/en (accessed on 6 March 2020).

195. Farabet, C.; Poulet, C.; Han, J.Y.; LeCun, Y. CNP: An FPGA-based processor for Convolutional Networks. In Proceedings of the 2009 International Conference on Field Programmable Logic and Applications, Prague, Czech, 31 August–2 September 2009; Volume 1, pp. 32–37.
196. Farabet, C.; Martini, B.; Corda, B.; Akselrod, P.; Culurciello, E.; LeCun, Y. NeuFlow: A runtime reconfigurable dataflow processor for vision. In Proceedings of the CVPR 2011 WORKSHOPS, Colorado Springs, CO, USA, 20–25 June 2011; pp. 109–116.
197. Chakradhar, S.; Sankaradas, M.; Jakkula, V.; Cadambi, S. A dynamically configurable coprocessor for convolutional neural networks. In Proceedings of the 37th Annual International Symposium on Computer Architecture, Saint-Malo, France, 19–23 June 2010; pp. 247–257.
198. Nuño-Maganda, M.; Torres-Huitzil, C. A temporal coding hardware implementation for spiking neural networks. *ACM SIGARCH Comput. Archit. News* **2011**, *38*, 2.
199. Chen, Y.; Luo, T.; Liu, S.; Zhang, S.; He, L.; Wang, J.; Li, L.; Chen, T.; Xu, Z.; Sun, N.; et al. DaDianNao: A Machine-Learning Supercomputer. In Proceedings of the 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, UK, 13–17 December 2014; pp. 609–622.
200. Du, Z.; Fasthuber, R.; Chen, T.; Ienne, P.; Li, L.; Luo, T.; Feng, X.; Chen, Y.; Temam, O. ShiDianNao: Shifting vision processing closer to the sensor. In Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, Oregon, 13–17 June 2015; pp. 92–104.
201. Intel® Neural Compute Stick 2 | Intel® Software. Available online: https://software.intel.com/en-us/neural-compute-stick (accessed on 16 April 2020).
202. Othman, N.A.; Aydin, I. A New Deep Learning Application Based on Movidius NCS for Embedded Object Detection and Recognition. In Proceedings of the 2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), Ankara, Turkey, 19–21 October 2018.
203. Get Started with Intel® Neural Compute Stick 2 | Intel® Software. Available online: https://software.intel.com/en-us/articles/get-started-with-neural-compute-stick (accessed on 16 April 2020).
204. Coral. Available online: https://www.coral.ai/ (accessed on 16 April 2020).
205. Examples | Coral. Available online: https://coral.ai/examples/ (accessed on 16 April 2020).
206. Hochstetler, J.; Padidela, R.; Chen, Q.; Yang, Q.; Fu, S. Embedded deep learning for vehicular edge computing. In Proceedings of the 2018 3rd ACM/IEEE Symposium on Edge Computing, SEC 2018, Bellevue, WA, USA, 25–27 October 2018.
207. Marantos, C.; Karavalakis, N.; Leon, V.; Tsoutsouras, V.; Pekmestzi, K.; Soudris, D. Efficient support vector machines implementation on Intel/Movidius Myriad 2. In Proceedings of the 2018 7th International Conference on Modern Circuits and Systems Technologies, MOCAST 2018, Thessaloniki, Greece, 7–9 May 2018.
208. Barry, B.; Brick, C.; Connor, F.; Donohoe, D.; Moloney, D.; Richmond, R.; O'Riordan, M.; Toma, V.; Nicholls, D. Always-on Vision Processing Unit for Mobile Applications. *IEEE Micro* **2015**, *35*, 56–66.
209. Liu, Q.; Huang, S.; Han, T. Demo: Fast and accurate object analysis at the edge for mobile augmented reality. In Proceedings of the 2017 2nd ACM/IEEE Symposium on Edge Computing, SEC 2017, San Jose, CA, USA, 12–14 October 2017.
210. Lee, S.; Son, K.; Kim, H.; Park, J. Car plate recognition based on CNN using embedded system with GPU. In Proceedings of the 2017 10th International Conference on Human System Interactions, HSI 2017, Ulsan, Korea, 17–19 July 2017.
211. Ezra Tsur, E.; Madar, E.; Danan, N. Code generation of graph-based vision processing for multiple CUDA Cores SoC Jetson TX. In Proceedings of the 2018 IEEE 12th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip, MCSoC 2018, Hanoi, Vietnam, 12–14 September 2018.
212. Rungsuptaweekoon, K.; Visoottiviseth, V.; Takano, R. Evaluating the power efficiency of deep learning inference on embedded GPU systems. In Proceedings of the Proceeding of 2017 2nd International Conference on Information Technology, INCIT 2017, Nakhonpathom, Thailand, 2–3 November 2017.
213. Chinchali, S.; Sharma, A.; Harrison, J.; Elhafsi, A.; Kang, D.; Pergament, E.; Cidon, E.; Katti, S.; Pavone, M. Network Offloading Policies for Cloud Robotics: A Learning-Based Approach. In Proceedings of Robotics: Science and Systems 2019, Freiburg im Breisgau, 22–26 June 2019.
214. Jana, A.P.; Biswas, A. Mohana YOLO based detection and classification of objects in video records. In Proceedings of the 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology, RTEICT 2018, Bangalore, India, 18–19 May 2018.

215. Chen, T.; Du, Z.; Sun, N.; Wang, J.; Wu, C.; Chen, Y.; Temam, O. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems—ASPLOS, Salt Lake City, UT, USA, 1–5 March 2014.
216. MNIST Handwritten Digit Database, Yann LeCun, Corinna Cortes and Chris Burges. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 16 April 2020).
217. National Institute of Standards and Technology | NIST. Available online: https://www.nist.gov/ (accessed on 8 January 2020).
218. Cook, L.T.; Zhu, Y.; Hall, T.J. Bioelasticity imaging: II. Spatial resolution. *Med. Imaging* **2000**, *3982*, 315–325.
219. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Journal of Machine Learning Research, 2010, 13th International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Sardinia, Italy, 13–15 May 2010.
220. TensorFlow Lite models | TensorFlow. Available online: https://www.tensorflow.org/lite/models (accessed on 8 January 2020).
221. Sequential—Keras Documentation. Available online: https://keras.io/models/sequential/ (accessed on 8 January 2020).
222. TensorFlow Lite inference. Available online: https://www.tensorflow.org/lite/guide/inference (accessed on 16 April 2020).
223. FlatBuffers: FlatBuffers. Available online: https://google.github.io/flatbuffers/ (accessed on 8 January 2020).
224. Magnitude-based weight pruning with Keras. Available online: https://www.tensorflow.org/model_optimization/guide/pruning/pruning_with_keras (accessed on 8 January 2020).
225. Python Imaging Library (PIL). Available online: https://pythonware.com/products/pil/ (accessed on 8 January 2020).
226. TkInter—Python Wiki. Available online: https://wiki.python.org/moin/TkInter (accessed on 8 January 2020).
227. NUCLEO-F746ZG—STM32 Nucleo-144 Development Board with STM32F746ZG MCU, Supports Arduino, ST Zio and Morpho Connectivity—STMicroelectronics. Available online: https://www.st.com/en/evaluation-tools/nucleo-f746zg.html (accessed on 15 March 2020).