Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

Ingegneria elettronica, telecomunicazioni e tecnologie dell'informazione

Ciclo XXXIII

Settore Concorsuale: 09/E3-ELETTRONICA

Settore Scientifico Disciplinare: ING-INF/01-ELETTRONICA

NEAR SENSOR ARTIFICIAL INTELLIGENCE ON IOT DEVICES FOR SMART CITIES

Presentata da: Gianmarco Cerutti

Coordinatore Dottorato **Supervisore**

Prof.ssa Alessandra Costanzo **Dr.ssa Elisabetta Farella**

Esame finale anno 2021

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Abstract

DEI - Department of Electrical, Electronic and Information Engineering "Guglielmo Marconi"
Electronics, Telecommunications, and Information Technologies Engineering

Doctor of Philosophy

Near Sensor Artificial Intelligence on IoT Devices for Smart Cities

by Gianmarco CERUTTI

The Internet of Things is in a continuous evolution thanks to new technologies that open the doors to various applications. While the structure of the IoT network remains the same over the years, specifically composed of a server, gateways, and nodes, their tasks change according to new challenges: the use of multimedia information and the large amount of data created by millions of devices forces the system to move from the cloud-centric approach, where most of the processing is delegated to the server, to the thing-centric approach, where also the node partially processes the information. Computing at the sensor node level solves well-known problems like scalability and privacy concerns. However, this study's primary focus is on the impact that bringing the computation at the edge has on energy efficiency: continuous transmission of multimedia data drains the battery, and processing information on the node reduces the amount of data transferred to event-based alerts. Nevertheless, most of the foundational services for IoT applications are provided by Artificial Intelligence. Due to this class of algorithms' complexity, they are always delegated to GPUs or devices with an energy budget that is many orders of magnitude more than an IoT node, which should be energy neutral and powered only by a tiny energy harvester. Enabling AI on IoT nodes is a challenging task. In this thesis, we tackled it from two sides. From the software side, this work explores the use of the most recent compression techniques for Neural Networks, enabling the reduction of state-of-the-art networks to make them fit in microcontroller systems. From the hardware side, this thesis focuses on hardware selection. It compares the latest AI algorithms' efficiency running on both well-established microcontrollers for Digital Signal Processing and state-of-the-art processors for machine learning applications. An additional contribution towards energy-efficient AI at the edge is the exploration of custom hardware for acquisition and pre-processing of sound data from the environment, analyzing the data's quality for further classification. Moreover, the combination of software and hardware co-design is the key point of this thesis to bring AI at the very edge of the IoT network. In the first chapter, the topic of near-sensor processing and artificial intelligence is discussed. Then, the thesis presents application scenarios with various multimedia sensors, and it demonstrates the capability of neural networks in these scenarios. Therefore, in chapter 3 and 4, this study explores different compression techniques to port complex neural networks on microcontrollers. Finally, chapter 4 and 5 describe the importance of hardware selection in the processing unit and sensors following the in-sensor processing paradigm.

Acknowledgements

First of all, I would like to thank my supervisor, Elisabetta Farella. She tried her best to make a good outcome out of this Ph.D. She guided me towards the right direction since the beginning and she puts a valuable effort in making the E3DA group cohesive and in leading a team atmosphere.

Despite her numerous appointment, she managed to find time for me, even in the late evening or on a train during her trips for holidays. I thank Elisabetta for showing me great interest and appreciation for my work. This was a great motivation for me.

I thank Fondazione Bruno Kessler, that has given me all the materials I needed. Moreover, it is a prosperous environment for research and I had many collaboration with other groups, like the Speechteck group, from which I would like to thank Alessio Brutti.

The E3DA was a great boost for me, starting from Davide Giovanelli, which still is an example for me about his impressive skills in problem-solving. I have found Davide inspiring because of his practical approach, transferring to me the ability to discern what is possible to make and what is not.

Oppositely to Davide, Francesco Paissan is a great example and inspiration for creativity, that I realized to be the most important skill during a Ph.D.

From the beginning, Giulio Gaio shows his patience in my preliminary stage of my Ph.D. When I started producing some results, he has appreciated a lot my work, and I found in this a great motivation to go forward.

I really thank Elia Leoni, Amy Murphy, Rajeev Pijare, Enrica Loria and Prabhakar Kumaravel for being always available when I have needed them and I have found their feedback a source of inspiration for my Ph.D.

For what concerns my period in the ETH Zurich, I thank Lukas Cavigelli and Renzo Andri. They showed me a very efficient methodology, they go deep in the details of things without leaving any choice to chance.

Finally, I would like to thank again all the people named here for making this Ph.D. a social and enjoyable experience.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
BNN	Binary Neural Network
CNN	convolutional Neural Network
DCT	Discrete Cosine Transform
DL	Deep Learning
DMA	Direct Memory Access
DNN	Deep Neural Network
DSP	Digital Signal Processing
FC	Fabric Control
FFT	Fast Fourier Transform
GIC	General Impedance Converter
GPIO	General Purpose Input/output
GRU	Gated Recurrent Unit
HWCE	Hardware Convolution Engine
IoT	Internet-Of-Things
IPS	Instructions Per Second
ISA	Instruction Set Architecture
KD	Knowledge Distillation
KWS	Keyword Spotting
MAC	Multiply-ACcumulate
MCU	MicroController Unit
MFCC	Mel-frequency Cepstral Coefficients
MIPS	Million Instructions Per Second
MOPS	Million Operation Per Second
MSE	Mean Square Error
NN	Neural Network
PDF	Probability Density Function
PULP	Parallel Ultra Low Power
RNN	Recurrent Neural Network
RTC	Real-Time Clock
SDK	Software Development kit
SED	Sound Event Detection
SIMD	Single Instruction multiple data
SoC	System On-Chip
SQNR	Signal to Quantization-Noise Ratio
STFT	Short Time Fourier Transform

Chapter 1

Context Information Extraction for Smart Cities

1.1 Internet of Things and Smart Cities

1.1.1 Background

The Internet of Things (IoT) is a multidisciplinary area based on the vision of a society where many of the objects that surround us will be networked and connected to the Internet offering innovative and efficient services [7]. The IoT initiated with RFID technology, which is increasingly used in logistics, pharmaceutical production, retail, and diverse industries [6]. The emerging wireless sensory technologies have significantly extended the functionalities of devices embedding them and therefore, the original concept of IoT has been extended soon to ambient intelligence and autonomous control. To date, not only RFID and NFC, but a number of technologies are involved in IoT, such as wireless sensor networks (WSNs), barcodes, intelligent sensing, low energy wireless communications, cloud computing [60, 42, 18].

This has clear consequences on the technology market. The McKinsey's report on the global economic impact of IoT [53] predicts that this impact in 2025 would be in the range of $2.7 to $6.2 trillion. These expectations imply the tremendous and steep growth of the IoT services, their generated data, and consequently their related market in the years ahead [61]. IoT opens great opportunities by enabling the collection of data from the environment, people, and daily life scenarios and the actuation of feedback based on real-time data processing [19].Thus, it affects a wide range of application fields, spanning from agriculture to automotive, smart cities, consumer devices, retail, manufacturing, supply chain, and many others [4]. Waste management can be more efficient by sensing the actual demand, and pricing based on actual consumption habits, thus encouraging virtuous behavior. Occupant recognition permits to adjust lighting, sound, air conditioning/heating so that occupants do not need to "push any button". Similarly, distributed audition permits to develop situational awareness, build real-time noise urban maps to mitigate noise pollution at critical times, and localize noise events for safety assurance. Living Labs are appearing as an emerging concept in which companies, public authorities, and citizens work together to create, prototype, validate and test new services, businesses, markets, and technologies in real-life contexts, such as cities, city regions, rural areas, and collaborative virtual networks between public and private players [65, 13].

1.1.2 IoT Infrastructure

Figure 1.2 shows that the IoT is structured into three tiers of devices. At the bottom, **IoT nodes** perform sensing and interact with the physical world. To assure scalability and ubiquitous network access, **gateways** collect, protect (under users' control) and route data from

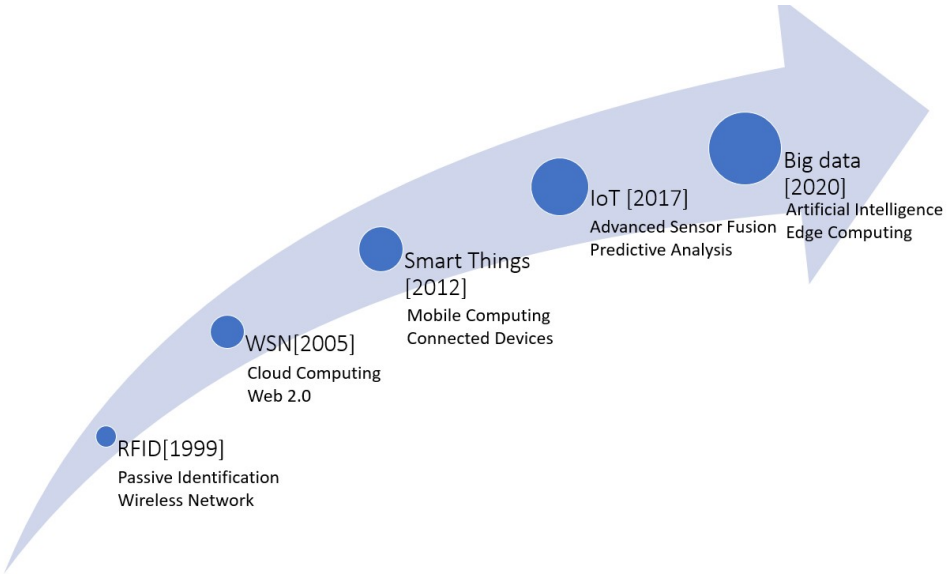

FIGURE 1.1: Timeline of IoT evolution [49, 61]

several and physically proximal IoT nodes, and route it to **servers**. The latter perform data aggregation and knowledge extraction and deliver physically enhanced cloud services. Some additional intermediate levels of aggregation might be needed, depending on the amount of data generated, the area covered by a sub-network, and the density of IoT nodes, among the others.

Due to their large number and ubiquity, IoT nodes need to be untethered and hence their power budget is very small, and is as low as sub-mW for miniaturized systems powered by energy harvesters. Due to their larger size and lower density, concentrators are expected to be mostly tethered, and hence their power can be much larger (e.g., in the order of Watts). A server blade dissipates a power that is two orders of magnitude larger.

1.1.3 Technology Challenges

The IoT provides many new opportunities to the industry and end-user in many application fields. Currently, however, the IoT faces many challenges in the fields of privacy, ethics, business model, global cooperation, and standards [95]. For what concerns technology issues, the two main points are:

- **Energy self-sustainability**: Energy will be a major technological challenge in the next five to 10 years; systems should be able to harvest energy from the environment and not waste any under operation [95]. In the perspective of implementing a pervasive network of untethered and autonomous devices, an IoT end-node requires being powered by batteries or energy harvesters. Depending on the application environment, the battery recharge or replacement may be prevented for technical reasons or, however, should occur with a low frequency to keep the maintenance costs contained (e.g. once every one or more years) [77].

- **Scalability**: specifically managing the large amount of data generated by the growing number of connected devices. Efficient exploitation of data becomes crucial to avoid overwhelming the network and, at the same time, guaranteeing a suitable service level. [19]

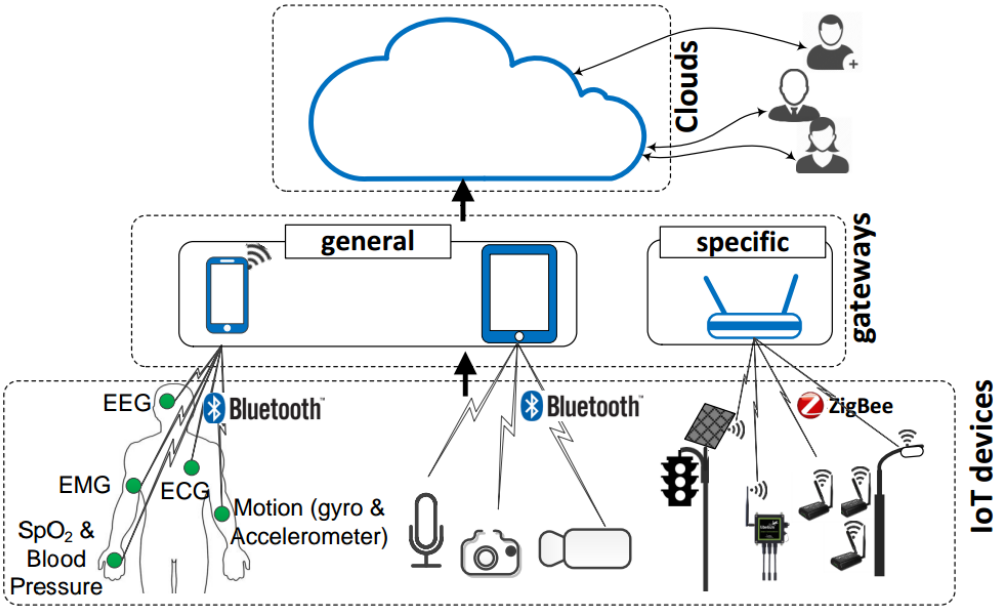

FIGURE 1.2: IoT infrastructure with different hardware layers, namely IoT node, Gateways and Cloud [61]

1.1.4 IoT end Node

IoT nodes perform sensing and interact with the physical world. They are the building blocks for the IoT, and they stand below the overall infrastructure. IoT nodes need to be energy autonomous and rely on battery and an energy harvester as energy source [4], because battery replacement is not considered in many cases due to the volume of devices connected. As opposed to purely battery-powered systems such as smartphones or laptops, energy harvested IoT nodes can operate perpetually, as long as the harvester power exceeds the average power consumption of the node, and can hence indefinitely sustain the power required by the IoT node. Fig. 1.3 depicts the general architecture of a sensor. An analog conditioning circuit is placed between the sensing element and the digital interface. Alternatively, an external ADC converter can be employed to digitize the sensor output. The radio interface communicates with the remote server through wireless communication, while the power management unit takes the power from the energy harvester and redistribute it over the node. Finally, the sensor signal is periodically sampled, according to the Nyquist sampling rate, and transferred to a central collector, i.e. the digital processor, which applies signal processing and filtering routines to the sensed data [19]. To reduce the power consumption of the overall IoT node, the microcontroller can run more advanced data processing to extrapolate the relevant information. In this way, the power consumption of the microcontroller decreases, but the system transmits fewer data, and the overall power consumption decrease. This approach of moving intelligence at the node level will be discussed in deep in the next chapter.

1.2 Near Sensor Processing

1.2.1 Problems of Cloud Computing

In many IoT applications, the user is not interested in the data stream itself, but only in specific events or some extracted information from them. For heart rate monitoring, the application can be interested only in unusual patterns of the patience to call an aid quickly.

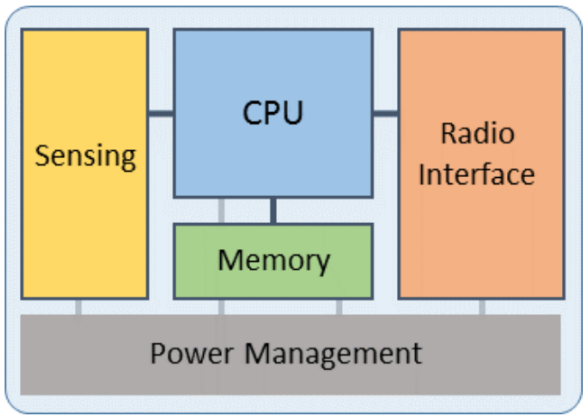

FIGURE 1.3: Main building blocks IoT end nodes

The same is in fall event detection, where the inertial data are not useful in a raw format, but processed to identify an event to trigger an alert. Audio event detection sends an alarm to the users when an intruder is detected in the proximity of his apartment. Finally, counting the number of cars crossing a street does not require the full video stream, but only the transmission of the counting index.

As already mentioned in the previous section, the IoT node is responsible for interfacing with the physical world. In a cloud computing paradigm, remotely located computing facilities (servers) are utilized using the Internet to store, gather, manage, and process the data coming from the node, that sends all the data to the remote server [2, 27].

Opposite to cloud computing, in a different computing paradigm, the IoT node not only acquires the physical data from the environment, but enables the data processing within the local system, transmitting only relevant information and alerts. This approach has different names in literature: mist computing [102], edge computing [72] or near sensor (edge) processing [44].

When the node has enough processing capability and energy budget for implementing this second paradigm, there are many reasons for which edge/mist computing is preferable to cloud computing.

- **Bandwidth** The bandwidth required depends on the application: simple applications like temperature room monitoring does not require a large data transfer. For such sensors, the duty cycle reduces the average data rate down to hundreds of tens of bits/second. Other kinds of sensors, such as accelerometers, microphones, or cameras, try to catch dynamic and transient events, so that the application needs high granularity data. For these applications, the bandwidth is in the order of kB or even MB in the case of video application (see figure 1.4). Therefore, computing at the edge solves the bandwidth problem of sending bulk amounts of data to the central server.

  Problems related to bandwidth are mainly saturation of the communication channel and power consumption due to the transmission of data. Hybrid approaches between edge and cloud computing, such as fog computing [10], solve the saturation issue thanks to a distribution of the computation over gateways, but it does not solve the power consumption problem, because the IoT node has to send all the data to the gateway.

- **Privacy** IoT implementation raises major concerns regarding individual privacy due to the ubiquitous connectivity of IoT devices and sensors [28]. Data can be collected from every connected appliance in every home and aggregated from every device in a

smart city. Privacy issues arise due to the streaming of sensitive data to a remote site, which can be used to identify and influence people [30].

Edge processing can limit the raw data to the IoT node and preserve the privacy of the user: If the IoT node just sends the event related to the physical phenomena, there is no way to identify the user where processed data are aggregated. Intuitively, sending the number of people present in a city area preserves privacy much more than sending the raw video to the cloud.

- **Centralization** Compared with the huge and centralized servers as found in cloud computing, edge computing has smaller edge servers that are distributed. One side effect is to reduce the probability of a single point of failure. For what concerns latency, it has been highlighted that cloud computing is suitable for delay-tolerant and complex data analysis, whereas edge computing is suitable for low-latency real-time operations [82]. Cloud computing provides also a solution for handling big data and processing them. However, it requires massive data storage volume, huge processing resources to deliver a high quality of service (QoS) in real-time [47].

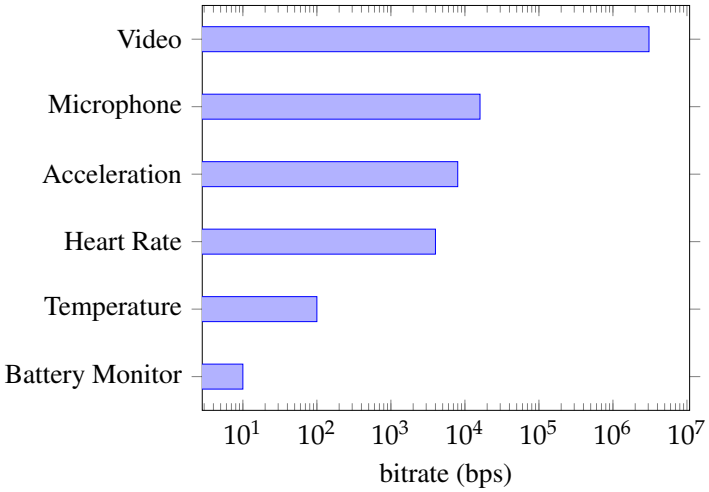

FIGURE 1.4: Bit rates for different kind of sensors [4].

1.3 Artificial Intelligence for IoT

In recent years, many IoT applications arose in different vertical domains, i.e., health, transportation, smart home, smart city, agriculture, education, etc. The main element of most of these applications is an intelligent learning mechanism for prediction (i.e., regression, classification, and clustering), data mining and pattern recognition or data analytics in general [61].

IoT applications are heterogeneous but there are recurrent foundational services. In other words, different application scenario often shares similar tasks to be computed starting from the data. Figure 1.5 shows many typical IoT applications that share a set of foundational services.

In all these three tasks, Artificial Intelligence and Deep Learning are recently getting the best performances, thanks to the increasing amount of datasets and computational resources. Residual Networks [32, 78] are examples of SoA for image classification. Deep neural networks outperform Gaussian Mixture Model (GMM) and Hidden Markov Model

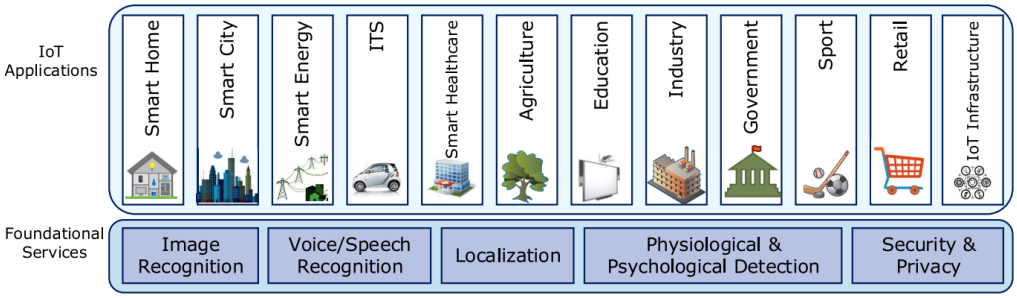

FIGURE 1.5: IoT applications and the foundational services [61]

(HMM) system in speech recognition [34, 29]. Literature demonstrates that traditional machine learning approaches are less effective than DNN for posture analysis in fall detection [20].

Focusing on Smart Cities applications, scene understanding, and context analysis are among the application domains where processing at the edge can be crucial. They often rely on computer vision. However, the combination with audio processing can highly improve the accuracy of event detection and activity recognition, complementing vision where line-of-sight occlusions or environmental light changes occur [97]. Furthermore, the use of audio detection alone can partially solve privacy concerns of camera-based solutions.

Thus, Sound Event Detection (SED) is a powerful tool for many applications such as traffic monitoring [64], crowd monitoring [55], measurement of occupancy levels for smart and energy-efficient buildings [94], and emergencies detection [24].

SED, as well as acoustic scene recognition, can benefit from understanding events locally where they happen both in terms of privacy [51] and reaction time, which can be kept in the range of ms. More than this, device lifetime can be guaranteed up to several years of operation when energy harvesting is applied and the transmission is limited to a few bytes.

Another common example of Artificial Intelligence in everyday life is Keyword Spotting (KWS): KWS in the audio domain targets to distinguish a specific small set of words in a predefined dictionary. This task becomes increasingly popular in numerous applications, like wake-word detection or conversational human-technology interfaces. The system cannot transmit everyday audio stream due to privacy concerns, latency, and energy consumption. Therefore, many fully voice-based interfaces rely on the user to preface their command with a keyword, that is processed on the device. The following speech translation and natural language processing are delegated to the cloud.

Artificial Intelligence and edge computing are strictly related because most of the application that uses multimedia data gets most of the benefit by local processing. Processing data locally preserves privacy and reduces the bandwidth transmission significantly, since multimedia data are the one with the highest throughput.

However, this always-on and battery-powered application scenarios for smart devices impose constraints on hardware resources and power consumption. Therefore, high complexity algorithms from state-of-the-art research such as deep convolutional neural networks cannot be used as they are (or off-the-shelf). In the next section, I will describe how Artificial Intelligence can coexist in the IoT end-node with some relevant examples.

1.4 From Server to Microcontrollers: Advances in Artifical Intelligence on Board

1.4.1 Trend In Artificial Intelligence for IoT

The machine learning field has seen a veritable breakthrough avalanche within the last few decades, driven by the extensive newly available compute capability, public accessibility to large and diverse datasets, and easy-to-use deep learning frameworks like Tensorflow, Torch, Caffe. Especially, Convolutional Neural Networks (CNNs) or Deep Neural Networks (DNNs) have revolutionized computer vision and data analytics in a broad spectrum of applications and challenges [68].

State of the Art neural network models require hundreds of watts for inference, hundreds of megabytes, and billions of complex floating-point operations. Canziani et al. [11] concluded that accuracy and number of operations are in a hyperbolic relationship. It means, that best performing models require more operations, as shown in figure 1.6. As a consequence, a current trend is to use networks of continuously increasing size and complexity because they generalize better than shallower ones.

An example of this concept is Sound Event Detection (SED), that I have already mentioned in the previous chapter as one of the foundational services for IoT. SED is a rather challenging task, especially when applied in outdoor contexts. After some pioneering efforts [93, 109, 58], which have not led to established solutions, recent progresses in deep learning and the release of sound event datasets and challenges like *UrbanSound8K* [81], AudioSet [22], ESC50 [71] and DCASE [56, 57] have reawakened interest in these applications, considerably improving the performance and paving the way to further developments. Nevertheless, advances in terms of accuracy and robustness of current acoustic event detection algorithms are achieved by using large neural networks, which are increasingly hungry in terms of computational power and memory: for instance, deep neural networks (DNNs) for SED such as L3 [16] and VGGish [33] require approximately 4M and 70M parameters, respectively.

In the computer vision domain, the VGG-16 [87] network contains about 140 million 32-bit floating-point parameters and it achieves 92.7% top-5 test accuracy for image classification task on ImageNet dataset. The entire network needs to occupy more than 500 megabytes of storage space and perform 8 GMACs in floating-point. [73].

As a consequence, the most recent and advanced solutions may be not practical if limited computational resources are available, as in the case of IoT application contexts. This prevents the development of applications for distributed monitoring in public spaces, which require a pervasive network of energy-neutral devices composed of cheap, low-power, low-complexity platforms. Enabling advanced machine learning on IoT nodes is, therefore, of great interest and is becoming an attractive research topic for a variety of digital signal processing applications. As a confirm, an entire new scientific community has formed around the topic of machine learning on these devices, namely the TinyML movement. TinyML devices come with tight restrictions in power, energy, memory, and compute capabilities, which is totally orthogonal to the requirements of the trend of new state-of-the-art ML algorithms.

To do so, different strategies exist, all of them target the power consumption reduction affecting as little as possible the classification accuracy. Here is a comprehensive list of the methods in literature:

- **Optimized Network Topologies**: Expedients to reduce the complexity of the model. For example, the authors of [79] test different architectures to explore the consequence of changing the size of different layers. By replacing the fully connected layer of an existing convolutional neural network (CNN) with average max-pooling, Meyer et al.

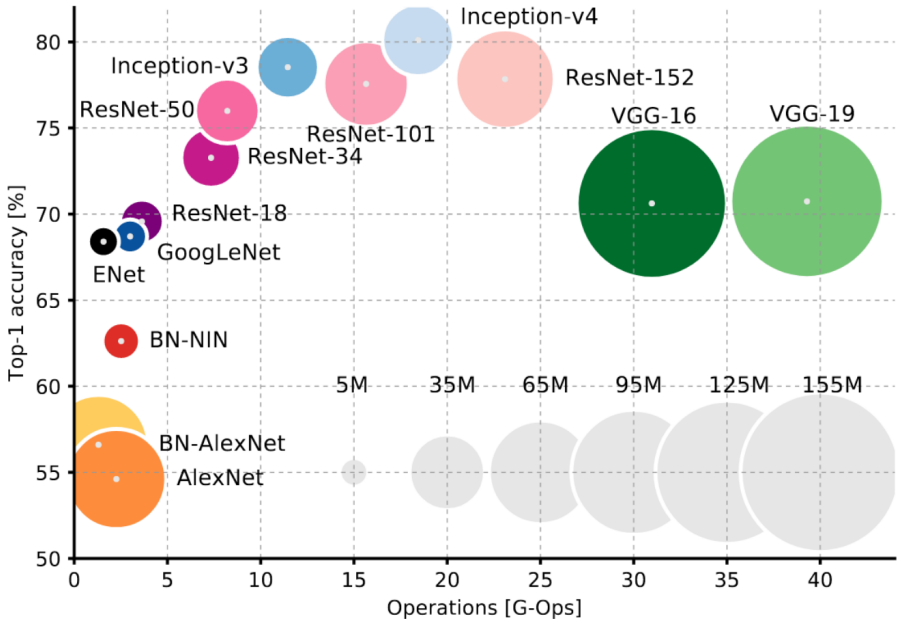

FIGURE 1.6: Top-1 accuracy in the ImageNet challenge versus amount of operations required for a single forward pass. The size of the blobs is proportional to the number of network parameters [11]

[59] reduced the number of parameters while increasing the accuracy for the targeted dataset. More recently, the authors of [39] achieves the same performance of AlexNet with 50 times fewer parameters reducing the input channel of convolutional networks thanks to 1x1 convolutions. Mobilenets [37] and Efficientnets [91] are network topology with a number of operations that can be controlled by some hyperparameters, and those change some feature of the network like the number of filters, input resolutions, and more generally the network's structure.

- **Network Pruning**: Neural networks are usually over-parameterized with significant redundancy in the number of required neurons. This results in unnecessary computation and memory usage at inference time. One common approach to address this issue is to prune these big networks by removing less relevant neurons and parameters while maintaining the accuracy [8, 31].

- **Knowledge Distillation**: This method takes an existing model, and it trains a new network with a smaller memory footprint, but that effectively mimics the original one. This approach is also referred to as Student-Teacher because the smaller network (student) is trained to get the knowledge and mimic the output of the larger one (teacher) [83]. The underlying idea is that the output of the neural network (soft labels) is more abundant in information than the hard labels and makes the training easier [35].

- **Approximate Computing**: Deep learning frameworks train neural networks using floating-point 32-bit weights and activations. However, neural networks are robust to input noise by design; thus, researchers have tried to reduce numerical precision to save memory, computation, and internal data bandwidth. This process, called quantization, aims to reduce the storage and computational costs of the inference task [50, 45]. On the other hand, it does not come for free: performance of the network in terms of accuracy can degrade. If properly trained, the weight-quantized networks can achieve performance close to the floating-point original models also on complex

classification tasks [48]. Zhou et al. [107] present a technique for lossless weight quantization down to 2 bits. This saves memory and bandwidth, but keeping activations in full precision requires always floating-point computations and 32 bit for each activation. Unfortunately, activations are more sensitive to quantization in terms of accuracy drop. The approximation of activations is quasi-lossless down to 8 bit [105] and it allows the hardware to parallelize 4 computations over a 32-bit register. As an extreme case of quantization, Binary Neural Network (BNN) reduces the precision of both weights and neuron activations to a single-bit [15, 75]. BNNs work on simple tasks like MNIST, CIFAR-10, and SVHN without any drop in accuracy [38]. On the challenging ImageNet dataset, BNNs have a drop of 12% [108, 89]. The gain from this accuracy degradation performance is an improvement of computation efficiency because binarization reduces the amount of memory required and compress 32 MAC operations in just two operations. Moreover, Binary operations convert multiplications in hardware-friendly `XNOR` operations. However, quantization is not always the solution when facing very deep models: the complexity reduction can be up to 32 times fewer parameters, and one of the previous technique should be taken into account when the application requires further levels of compression.

1.5 Advances in Hardware For Very-Edge Computing

1.5.1 Microcontrollers Directions

The interest in Artificial Intelligence on IoT nodes have awakened primary digital electronic industries.

GPUs and highly parallel architectures achieve the fastest inference and have enough memory to run state-of-the-art models. On the other hand, their power consumption limits the use of these architectures to big servers.

At the microcontroller lever, where the system can be powered only by an energy harvester, there are many architectures tailored to signal-processing. ARM has more than 180 billion processors shipped worldwide thanks to the ample range of cores targeting different applications. While ARM Cortex-A families targets high-performance devices like mobile phones, the leading commercial architecture for embedded devices is the ARM Cortex-M series [100]. M0 and M3 cores aggressively reduce cost and power consumption, offering limited computational capabilities but adequate in many IoT-like applications. Of great interest for signal processing applications is the Cortex-M4 family that provides hardware and software support for Digital Signal Processing (DSP), SIMD, and MAC instructions. Finally, Cortex M7 cores are the best performing in terms of execution time, but they are meant to work with a fast clock and therefore higher power consumption than the earlier families.

For the M4 and M7 families, ARM released the CMSIS libraries, which make use of SIMD instructions and simplify programming interfaces, thus reducing development effort. On top of it, CMSIS-NN is the official support for Neural Network inference functions, and many tools exist to export neural networks from the most popular frameworks to CMSIS-NN API.

In February 2020, ARM launched a new family of Cortex-M core, namely M55, which proposes ad-hoc architectures for Artificial Intelligence. Thanks to Vector Processing Technologies, they improve of 15x the performance on Machine Learning algorithms compared to previous Cortex-M solutions. The Cortex-M55 can be paired with an Arm Ethos™-U55 Neural Processing Unit, specifically designed to improve machine learning inference [101]. In the future, microcontroller industries will produce devices that equip these cores. Unfortunately, at the time of this thesis, they were not available.

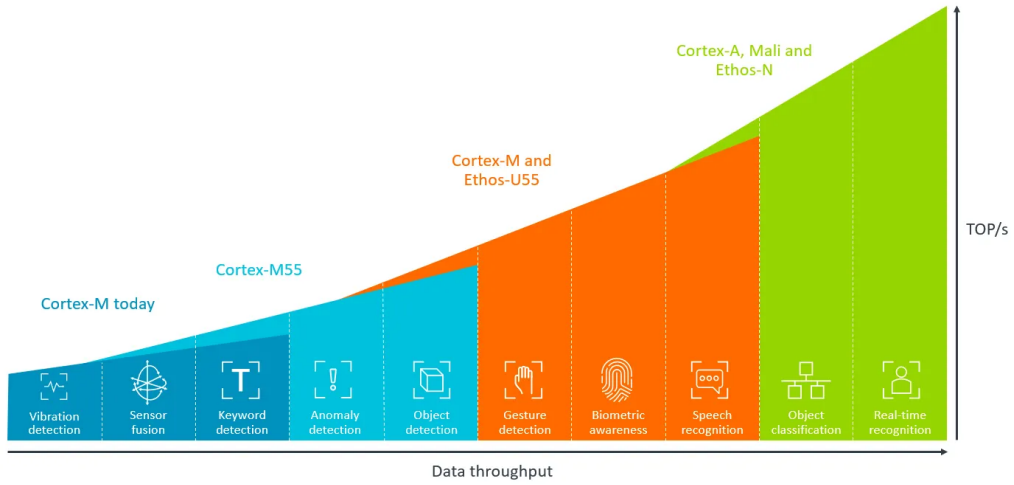

FIGURE 1.7: Different ARM core families and their related application scenario [1]

The world of artificial intelligence is in continuous evolution; therefore, hardware and instruction sets should adapt to the most recent innovation. In this context, ARM cortex cores are not always the most suitable solution since their instruction set cannot be modified for license reasons.

RISC-V ISA is getting popularity because, in contrast with ARM ISA, it allows the design of processors without a license fee and the designers can add ad-hoc instructions for specific neural networks model or layers. Following this trend, companies that have always included ARM core in their products are now exploring the use of RISC-V based architecture, like PolarFire SoC from Microchip or RV32IMC from NXP.

Recently, a novel parallel processor, based on the RISC-V ISA, has been launched [96]. GAP8 is a commercial processor, implemented from the Parallel Ultra Low Power (PULP) open-source project[1]. This processor has similar power requirements of the Cortex-M family (hundreds of mW) with up to 20 times higher computation performance for machine learning applications, thanks to near-threshold parallel computing [96]. Furthermore, it features RISC-V extensions providing accelerating the BNN processing. The `popcount` instruction boosts the processing significantly for BNNs and other quantized neural networks.

TABLE 1.1: Examples of embedded platforms and their hardware capabilities.

Board Name	Flash[KB]	RAM[KB]	Power [mW]	MIPS
Arduino uno [ATmega328]	32	2	60	20
ChipKit uc32 [PIC32MX]	512	32	181	124.8
STM32L476RG [Cortex-M4]	1024	128	26	100
TI MSP432P4111 [Cortex-M4]	2048	256	23	58.56
BeagleBone Black [Cortex-A8]	Ext	524288	2300	1607
Raspberry Pi 3 B+ [Cortex-A53]	Ext	1048576	5500	2800

1.5.2 Hardware Design for Artificial Intelligence

The previous chapter focuses on general-purpose devices, that allows products to be low-cost and off-the-shelf. The disadvantage of these devices is the loss in efficiency due to their flexibility. FPGA implementations still have the flexibility needed for new ML advances

[1] https://www.pulp-platform.org

and already significantly boost performance and energy efficiency. However, they are still too expensive for large-distribution of devices and are also known to be at least one order of magnitude worse than custom ASICs. Nevertheless, the higher energy efficiencies and throughput come with a price, which is the loss of flexibility to adapt to the very fast-changing AI research field. Furthermore, custom ASIC accelerators have very high non-recurrent costs, which can just be compensated with a high number of sold chips. After this clarification about the efficiency vs flexibility trade-off, peak throughput and energy efficiency are achieved by ASIC accelerators. There are hardware accelerators meant to make neural network inferences more efficient and faster. A known example is Movidius' Myriad 2, which computes 100 GFLOPs and needs just 500mW@600 MHz [62, 40]. However, they are still over the energy budget of IoT application, and therefore several dedicated hardware architectures have been proposed to improve energy efficiency.

For binary-weights neural networks, YodaNN is the only BWN accelerator in literature and achieves a core efficiency up to 61.2 TOPS/W [5]. Moving to Binary neural networks, BinarEye [63] and XNORBIN [3] achieve an energy efficiency of 115 TMAC/s/W and 102 TMAC/s/W respectively.

These examples target ultra-low power consumption processing of neural network for edge computing; they present stable accuracy and convincing memory footprint for their model, but they do not focus on the power consumption of the overall system. In fact, figure 1.3 shows that also sensing should be taken into account, as well as the sensor-to-processor data transfer, whose bandwidth linearly increases with the amount of produced data, determines an additional energy consumption, impacting the total budget.

For always-on multimedia applications, the sensing consumption can be dominant or comparable with the processing consumption. In a SED system, the microphone itself, its amplification circuitry, and the communication between the microcontroller and the sensor are a relevant part of the energy used for audio analysis. Chapter 4 presents an edge computing system for sound event detection where it is shown that the system uses more than 60% of the power budget for sensor reading and feature extraction, and only the remaining part is for classification.

To tackle this bottleneck, hardware designers present innovative sensors that have these two key points:

- Event-based alarms: The sensor communicates and send data to the processing unit only when there is a relevant event.
- In-sensor preprocessing: Data is partially preprocessed on the sensor chip and therefore data communication is reduced.

For embedded computer vision applications, a camera node with the described characteristics is labeled as "smart", indicating a system that is not only able to capture data, but also to process it thanks to a dedicated processing sub-system for data analytics. An example is the FORENSOR sensor, where the camera embeds on-chip an image processing algorithm for real-time event detection, exploiting a dynamic background subtraction approach to identify the so-called hot-pixels, i.e. pixels with an intensity variation, and possibly corresponding to a non regular action [46]. Paissan et Al. [66] evaluates the use of the FORENSOR sensor in a classification scenario. They achieve 87% accuracy in people/car classification, proving that the overall system consumes less than 2mW, thus being adequate for an IoT scenario.

Similarly, in the audio processing domain, intelligent hardware components, such as a Mel-spectrogram extractor or a speech enhancer, can support the extraction of events or patterns with optimized energy efficiency and performance. For example, audio classification systems are often based on the regular sampling of microphone inputs with frequencies in the range from 16 kHz to 192 kHz, which fixes a minimum power consumption that cannot

be avoided. Thus, a "smart" microphone returning only features for classification instead of the raw data stream would be beneficial to the energy efficiency of the whole system. In fact, the following machine learning task will be activated only when relevant information has to be processed, implementing an event-based paradigm.

In this direction, Mayer et al. [54] designed an analog front end that extracts time-frequency features in hardware: it has a microphone, a bank of filters and an amplification circuit, all designed targeting low power consumption. In this way, the authors move the focus of optimization also in data acquisition and preprocessing. The output is a pulse of a certain time duration that indicates how long a specific frequency is present. It has a great benefit versus a normal sampling approach: the front end rises a pin when the specific frequency band is present. Therefore, the system acquires only the pulse, so that the microcontroller reads the data only when something interesting for the system is going on, following the event-based acquisition paradigm. The author of the analog front end presents an underwater monitoring application, which is relatively simple and for this reason, the classifier is not powered by artificial intelligence. In chapter 5, I will present a keyword spotting system that processes audio data and distinguish 10 different keywords in the speech: using parallel filters, the microcontroller is able to reconstruct spectrogram-like information, which is passed through the BNN for further classification.

1.6 Thesis Contribution

The thesis's structure is the following: Chapter 1 has described the importance of IoT in the present historical moment, the main challenges, and how near sensor processing can convincingly address the technological problems of cloud-based solutions. Next, I have presented the relevant link between the Internet of Things and Artificial intelligence, showing that Neural Networks are the basic processing engine for knowledge extraction from most IoT data. Then, I focused on Artificial Intelligence, presenting the growing trends in accuracy and memory size, and computation complexity. A review of complexity-reduction algorithms has shown that the problem does not have a unique solution. Next, a review of hardware advances in the field of near sensor processing follows: after presenting commercial and more advanced microcontroller, the focus has moved to hardware accelerators and finally to mixed sensing and processing approaches, where the hardware also optimizes acquisition and pre-processing of data.

Chapter 2 is one of the near-sensor processing prototypes in this thesis, targeting outdoor people detection. To the best of my knowledge, this is the first study that explores the use of a thermopile array in outdoor conditions. The low-resolution thermal images provided by this sensor make the processing compatible with microcontrollers. Here is my first attempt to use neural networks for multimedia data. Compared with more traditional computer vision techniques, artificial intelligence is the best performing for the specific setup.

Chapter 3 moves to another multimedia application, known as Sound Event Detection. Here I face the complexity problem because state-of-the-art neural networks in this task are not suitable for near sensor processing. Therefore, I explore the combination of extreme distillation (student-teacher approach) and quantization techniques. In this thesis, the student-teacher paradigm is stressed to obtain student networks many order of magnitude smaller than teacher networks. In this way, SoA networks can be used as teachers, and student networks respect a microcontroller's constraints.

Chapter 4 focuses more on the hardware side. Here, I prove the efficiency gain given by hardware that targets machine learning problems. Multicore capabilities are an acceleration factor for many neural networks, but most importantly, the combination with low voltage

threshold hardware achieves impressive computation efficiency. Moreover, popcount instructions are a crucial point to make Binary Neural Network more efficient. Finally, the overall system has a power budget that matches with an IoT end-node. The use of the particular hardware platform is the enabling point for the execution time and energy consumption. At the time of this thesis, binary neural networks were rarely used, and many application scenarios did not have a research example for such networks. Here I investigate the use of BNN for sound event detection, concluding that their efficiency definitely fits the artificial intelligence at the very edge. On the other hand, the accuracy drop due to the 1-bit resolution is still an open issue that prevents the deployment of such structures for real-application scenarios.

Until this point, the thesis presents significant advances in the compression of neural networks to enable their execution in IoT nodes, but the energy consumption does not include just processing. So far, the thesis presents mainly significant progress in the field of neural networks to enable their execution on IoT nodes. Nevertheless, strategies to limit power consumption do not only include acting on processing. A classification system also includes sensing and pre-processing, which can be energy-hungry, especially in multimedia applications. For this reason, Chapter 5 explores the use of custom hardware for audio sensing and pre-processing. Here, power reduction is given by the low power sensing system and the low communication bandwidth between the sensor and the processing unit. The custom hardware does not transmit the audio stream but only relevant features in a binary version. The processing algorithm, again a Binary neural network, is perfectly integrated with such kind of data. Finally, the Conclusion section presents a summary of the thesis and the lesson learned over the whole Ph.D. work.

To summarize, my thesis's contribution is moving AI-based computation (e.g., NN) at the edge, exploring novel hardware and software co-design solutions.

Chapter 2

Low Resolution Thermal Sensor for Outdoor People Detection

2.1 Overview

As anticipated in the introductory chapters, we applied our approach of bringing intelligence near the sensor, while considering energy efficiency both at hardware and firmware level, in specific application scenarios. In this chapter we apply our approach on a IoT node for presence detection. Presence detection is a useful functionality to make our living spaces smarter and is implemented through several kinds of sensors and smart devices. Nowadays, most of the research works focus their attention on the indoor environment. However, outdoor people detection is also relevant: in the smart city era, people detection systems can be useful in public spaces.

Therefore, we focus on outdoor people detection. This kind of scenario is more challenging than detection in indoor environments for many reasons, such as the high variability of several parameters, i.e. sudden changes of temperature and light due to weather conditions, or high variability of human and background temperature and wide daytime temperature variation. Typical solutions for outdoor people detection are based on sensors deployed in the environment, typically cameras. However, video surveillance can be invasive, not preserving citizen privacy, and generating a large amount of raw data, not all meaningful, to be analyzed. Hence, here we target a low cost, low power and resource-constrained system, to detect the presence in outdoor scenarios while preserving privacy. For these reasons, we explored a very low-resolution thermal camera, that generates easy to process images where people are indistinguishable. Moreover, The approaches and techniques chosen are not computationally intense and thus thought for a resource-constrained embedded system. The processing algorithms implemented can be divided into two blocks: 4 different well-established computer vision approaches and a Convolutional Neural Network. All of them have never been applied to thermopile array sensor input before. In this chapter, we present the first use of a thermopile array in outdoor environments for people detection. That is not trivial because outdoor conditions complicate the detection. To fit existing methods to our sensing device and environment we explore and test solutions that are innovative in the field of low-resolution thermal images. The whole study is then confirmed by a real-time implementation running the Convolutional Neural Network in a microcontroller, which shows that it is possible to achieve 76.7% of accuracy in the outdoor people detection task using 16.5 mW of power and 6 KB of RAM.

2.1.1 Related Work

People detection systems usually include camera based solutions, but the output of this kind of sensors is strongly illumination-dependent and requires a high computational load [74]. Furthermore, vision systems often limit privacy. Our application scenario requires the system

to work in a wide spectrum of illumination conditions, from night to direct sunlight. For these reasons, we deploy a system based on thermal spot detection.

Detection of people using infrared radiation is a broad research area. In this field, Passive InfraRed sensors (PIR) are a frequent solution for fire alarm and intruder detection since the eighties [99]. However, there are many disadvantages related to the physics of the measured process, which is in many cases affected by noise. Some examples are background radiation, reflection and change in ambient temperature [43]. For this reason, researchers adopting this solution focus on reducing the number of false recognitions. In the work presented in [36], the authors try to reduce the number of false alarm events using two different algorithms, based on the statistical characteristics of false and target alarms. Nevertheless, the main advantage of PIR technology is the low energy consumption and low cost of the product. To reduce the number of false alarms, many works combine information from different types of sensors, as in [12], where each sensor node includes a microphone, x-y axis magnetometers, and four passive infrared (PIR) motion sensors.

An alternative solution for detecting people outdoor is using thermal cameras. A thermal camera acquires a digital image and not an analog signal such as PIR sensors. From these images, it is possible to differentiate humans from other sources of noise, coming from the environment. For example, the authors of [17] have developed a processing technique for pedestrian detection. They implement conventional two-stages approaches to detect people in thermal imagery that combines a particular background-subtraction with a classifier. In their work, dataset is recorded with 320 x 240 Raytheon 300D thermal sensor at 30 Hz. It is 3 orders of magnitude more expensive than PIR and the huge amount of data requires a high-level processing platform. In addition, thermal images can show too many details and privacy is not always guaranteed.

The middle ground between PIR and thermal cameras are thermopile arrays, whose low resolution is a compromise between output information, power consumption and costs. Thermal array sensors can provide coarse-grained detection while preserving privacy of the subjects [26]. In this work, we adopted the Grid-EYE Panasonic thermopile array [26], a promising low-cost sensor that has been recently exploited also in [9] and [85]. The authors of [9] focus their attention on people counting indoor, with excellent results. On the other hand, their hypotesis of a fixed and stable temperature for both background and human body is not applicable to the outdoor conditions, considered in our work.

In [85] the system measures the speed of the gait using the Grid-EYE sensor. It detects the thermal peak using background subtraction technique very similar to one of the techniques presented in this document. The paper emphasizes the presence of noise in the environment, such as airflow or spontaneous change of air temperature. However, they do not explore the outdoor environment, even if their system is robust to noise.

A similar work is the one of Gomez et Al. [25], where the authors target edge computing implementing people counting using thermal images. They implemented a CNN on a microcontroller, analyzing the power consumption in a battery-powered prototype. In their case, the environment is indoor, therefore less affected by thermal noise caused by weather changing.

They reach a power consumption of 34.4 mW, which is close to our results. However, the execution time is $\sim$63 s, with a consequent gap between two collected images sufficient to miss detection of people crossing and standing for shorter periods. In our case, the execution time of $\sim$ 5 ms guaranteeing an higher temporal resolution.

2.2 System Description

2.2.1 Application Scenario

The present work has been conducted as contribution to the MAKE IT PAYT project, where the goal is applying sensors and RFID tags to optimize door-to-door waste collection. The waste bag are equipped with RFID tags to associate the user with the related waste, to implement the "pay-as-you-throw" paradigm. One request set in the project is to activate the RFID reader only when necessary. In particular when the garbage collector in front of the back of the truck, before she/he empties the waste into the vehicle and generate an event to activate the RFID antenna (see figure 2.1). Even if we focused on a specific application, it is representative of a generic mobile outdoor scenario, where background and environment conditions may present wide and sudden changes.

Thus, the conditions are:

- **Covered Area 2m x 1.4m:** the garbage collector remains in a small area behind the truck while emptying the bags.
- **Ambient Temperature Range from -10 C° to 40 C°:** possible range of temperature in urban environment.
- **Robustness to Light:** the system should work in wide range of light conditions, from dark at night to direct sunlight.
- **Criticality of False Negatives:** missing an emptying event is a fault, since the system does not turn on the RFID antenna.

2.2.2 Sensing Device

The second and third requirements lead us to choose a thermopile array as sensing device. The single thermopile measures infrared radiation from a small area. An array of them generates an heat-map. The specific device used in this work is the Grid-EYE sensor, an IR camera in a compact all-in-one SMD package [26].

The main specification are:

- 64 thermopile elements in an 8x8 grid format
- 10 Hz frame rate
- 30 mW power comsumption
- 60° viewing area
- IR detector and mixed signal processing IC in a single ceramic package

We locate the prototype at a 2.85 meter with a 60° angle, to cover a 3m x2 m detection area. The low number of pixel makes the hardware simple and fast.

2.2.3 Outdoor Scenario

For many reasons people detection in outdoor environment by means of thermal cameras, PIR or thermopile arrays, is more challenging than in indoor environment:

- Outdoor temperature is subject to frequent changes throughout the day, differently from what happens in close spaces, where environment has a fixed span of possible temperatures, tailored to the human comfort. Therefore, background modeling methods should consider this to be robust.

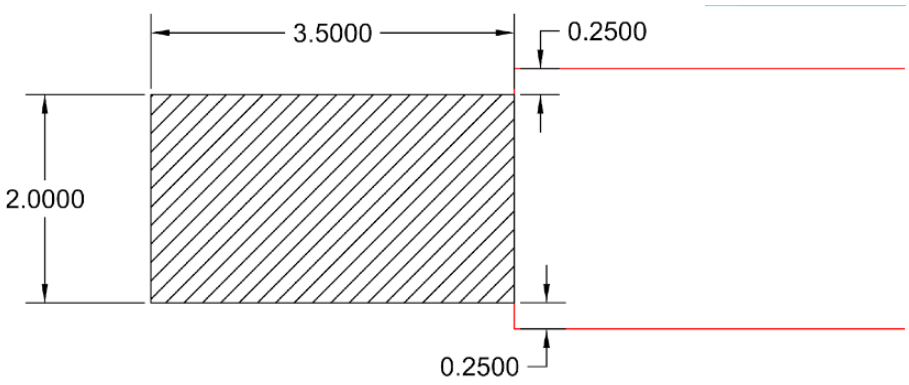

FIGURE 2.1: Area in which the operator should be detected. The truck is at the right.

- As a matter of fact, human body temperature in outdoor scenes depends on weather conditions. During the winter, warm clothes such as coats reduce drastically the surface temperature. As a consequence, infrared radiation decreases.
- In the daytime, soil heat-map may have different "red spots" due to sun lighting in part of the field of view. Shadows due to objects in the scene create the opposite result.
- Wind flows change drastically scene temperature in few seconds. This phenomenon is definitely negligible in indoor environment.

These elements differentiate the techniques implemented in our study compared to those used in indoor environment by means of Grid-EYE. For example, authors in [9] assume that background and humans have a fixed interval of temperatures. As a consequence, they detect people in correspondence of a set of pixel values above a fixed threshold. The assumption done in that work, as already explained before, is not applicable to outdoor environments. Comparing pixel temperature value with a fixed threshold is not a viable solution in our scenario. Finally, outdoor background model must adapt more quickly than the indoor one to possible immediate changes due to sudden heat flows (warm or cold).

2.3 Methodology

2.3.1 Classic Computer Vision Approaches

The continuous change of background explained in the previous section forces to implement processing techniques taking outdoor problems into account. For this reason, it is more convenient to obtain a relative value between the environment and the person, by means of background estimation and extraction of the foreground components or by means of comparison between consequent frames.

Thus, we compare the performance of four different techniques providing results on they accuracy to detect people by computing their F1 score. The four approaches are:

- Frame Difference (FD)
- Gradient Difference (GD)
- Background Correlation (BC)
- Background Variance Detection (BVD)

Figure 2.2 shows that only the first two algorithms pre-process the image by means of an exponential filter.

Frame Difference technique computes the absolute value of the difference between the input frame and the previous frame. The result is an 8x8 matrix and from this it extracts the maximum element. At this point, the value is compared with a threshold to classify the presence of an heat source. The equation that summarizes the algorithm is:

$$FD_{i,j}[n] = max(|I_{i,j}[n] - I_{i,j}[n-1]|) \quad \forall (i,j) \in image \tag{2.1}$$

where $I_{i,j}[n]$ is the (i,j) pixel in the current frame and $I_{i,j}[n-1]$ is the (i,j) pixel in the previous frame.

Gradient Difference is very similar to the previous one, but there is an additional step before the frame difference. For each frame, the algorithm computes the gradient magnitude between the pixel and the next ones. The gradient magnitude is calculated in two steps. First of all, the soble operator calculates the horizontal and vertical gradient for the whole image, pixel by pixel. Then the magnitude is computed considering it as a vector. The result is a new pixel for each one of the raw image, and overall a 8x8 matrix. Then it computes the gradient magnitude of the previous frame and finally it calculates the the absolute value of the subtraction between the two elements.

$$GD[n]_{i,j} = max(|GMag(I_{i,j}[n]) - GMag(I_{i,j}[n-1])|) \tag{2.2}$$

GMag is a function that takes a frame and generates a new 8x8 matrix. It is composed of the magnitude of gradient vector for each pixel. Using the gradient magnitude, the output is more sensitive to moving sources than the frame difference.

Background Correlation provides a more complex model for the background. It is computed by a weighted combination of several previous images. When the sensor is powered on, the background is initialized with the first frame. Next, each pixel of the background is updated using an exponential filter:

$$BG_{i,j}[n] = (1-\alpha)BG_{i,j}[n-1] + \alpha I_{i,j}[n] \tag{2.3}$$

$BG[n]$ is the background image at the step n and α is the exponential filter constant. The idea is to slowly adapt the background model based on the new frames. Following, for each new frame, the algorithm computes a cross-correlation between the current frame and the background. For each c x c window in the frame, the following equation (2.4) computes a scalar value between 0 and 1.

$$\rho_{i,j} = \frac{\sum_{(n,m)\in N} I_{n,m} * BG_{n,m}}{\sum_{(n,m)\in N} I_{n,m} \sum_{(n,m)\in N} BG_{n,m}} \tag{2.4}$$

N is a c x c sub-matrix centered in the pixel (i,j). The size of N, c, is considered as a parameter. When background and current frame are similar, the output matrix is filled with 1. When a heat source appears, the pixel of the output matrix relative to the foreground element approaches the value of 0. For the sake of clarity, values are mirrored with this equation:

$$BC_{i,j} = 1 - \rho_{i,j} \tag{2.5}$$

in this way obtaining a low value means that there are no foreground elements. Higher value corresponds to differences between current frame and background.

In *Background Variance Detection*, the scene model increases in complexity and contains both mean and variance of the pixel over time. Mean is calculated as 2.3, considering $\mu_{i,j} = BG_{i,j}$. The equation 2.6 describes a possible way to update the variance using the mean and

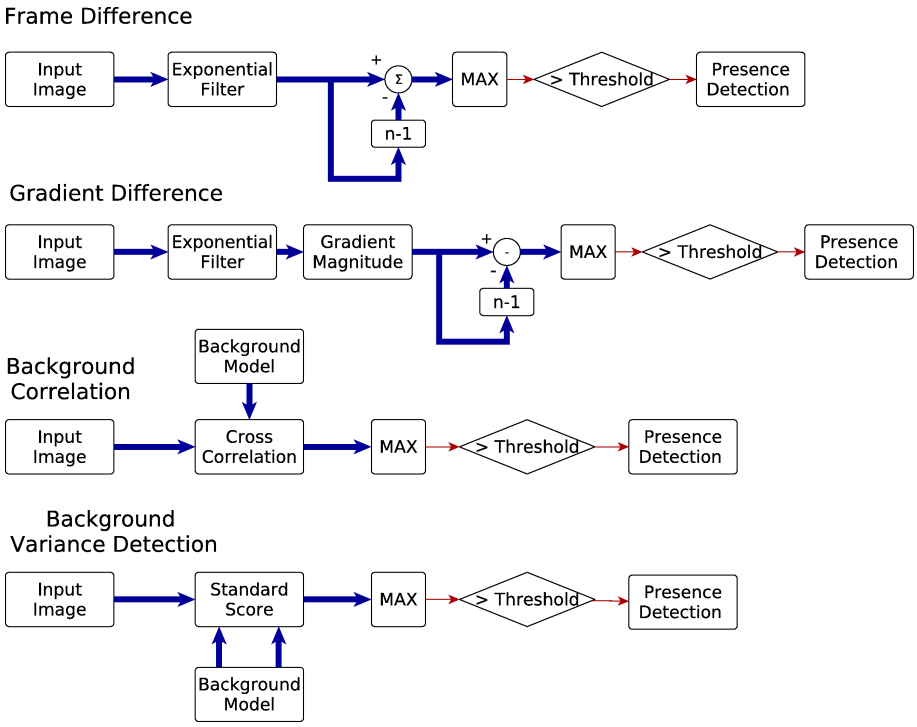

FIGURE 2.2: Computational graph of the algorithms. Blue bold arrow means 8x8 images, red light arrow means single values.

the current frame:

$$\sigma_{i,j}[n] = (1-\alpha)\sigma_{i,j}[n-1] + \alpha(\mu_{i,j}[n] - I_{i,j}[n])^2 \tag{2.6}$$

Next step is the comparison between the current frame and the background frame. The calculated value is as follows:

$$BVD_{i,j} = \left| \frac{\mu_{i,j} - I_{i,j}}{\sigma_{i,j}} \right| \tag{2.7}$$

The model is based on ideally fitting a Gaussian probability density function for each pixel [69]. The value computed, called standard-score or Z-score, is a distance between current data and mean, scaled by the variance.

Moreover, the algorithm does not update variance and mean if the output is greater than the threshold. This is because the foreground object should not be included in the background model.

Figure 2.2 shows the block diagram of the algorithms. The outputs are 8x8 matrices, but the system should generate an event when a person appears. For this reason, it takes the maximum value of the output matrix, then compares the result with a threshold in order to generate the trigger signal. The four algorithms perform in a different way, depending on the specific scene. Each of them has its own pros and cons.

Frame difference technique is very easy to implement, it is basically a subtraction frame by frame. The resulting hardware implementation has therefore real-time performance on most platforms. On the other hand, if a person stands still in the camera field of view she/he dissapears in the image, because there are no movements. As a matter of fact, calculating the frame difference corresponds to implement a motion detector and therefore static foreground elements are not detected.

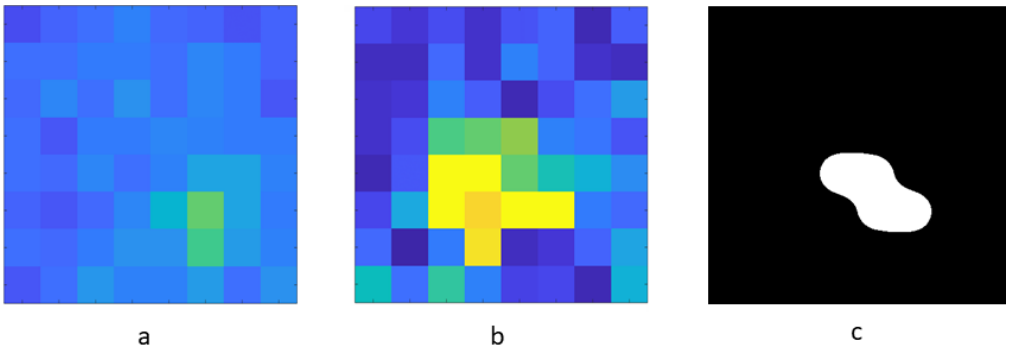

FIGURE 2.3: Data acquisition and processing while a person is in the field of view. From the left: (a) raw data (b) output matrix of gradient difference technique (c) output of the background variance detection after the threshold comparison for each element of the matrix.

Gradient difference approach is heavier from a computational point of view, but emphasizes the movement of a heat source. The result is an higher signal to noise ratio. However, it is not yet able to detect a non moving target.

Background correlation approach needs to compute the background model and the output matrix. To generate a single frame, the algorithm applies the equation (2.4) for each pixel, incrementing the computational demand. However, static objects can be detected but only for a short time. Equation (2.3) incorporates each element in the scene in the background. Foreground elements disappear after a certain number of frames. This number depends on the α parameter in equation (2.4). The same parameter tunes the ability to adapt the background to changes in the environment, providing a trade-off between static detection and background adaptation. If $\alpha = 1$, the background is the previous frame and the result is similar to the frame difference algorithm. If $\alpha = 0$, the background is the first frame and remains the same throughout the whole execution of the algorithm. In this case the foreground elements never disappear, but the algorithm is not able to adapt to changes in the environment.

Background variance detection is the only algorithm able to detect a non moving person. In fact, foreground elements are not included in background because mean and variance of the pixels over threshold are not updated. However, α is still a tunable parameter related to the changes of scenario.

Figure 2.3 shows the system working in an outdoor scenario. From raw data Fig 2.3.a is difficult to recognise that there is a person in the scene. In the output of gradient difference Fig 2.3.b the presence is more visible. In Fig 2.3.c the output of background variance detection is compared to a threshold. In figure 2.3, the shape of a walking person as seen from above is evident.

2.3.2 Convolutional Neural Network

We applied background subtraction before feeding the image to the neural network. This is because of the too wide range of possible temperature and to highlight the differences between the background and not only the absolute temperature value. In fact, a static sunspot can create an indistinguishable pattern from a low resolution image.

We implemented a running background average: the main advantage of this method is the adaptive maintenance of the background model while changes occur in the scene [88]. When the sensor is powered on, the background is initialized with the first frame. Afterwards, the algorithm updates each pixel of the background using an exponential filter:

$$BG_{i,j}[n] = (1-\alpha)BG_{i,j}[n-1] + \alpha I_{i,j}[n] \tag{2.8}$$

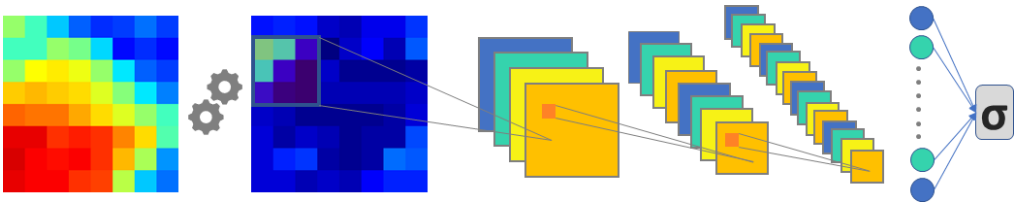

FIGURE 2.4: Flowing diagram depicts the raw image taken from sensor, then performing a running average background and then passed to a CNN

$BG[n]$ is the background image at the step n, i and j are the indexes of the image. The idea is to slowly adapt the background model based on the new frames.

Finally the input image is computed as following:

$$INPUT_{i,j}[n] = \mid BG_{i,j}[n] - I_{i,j}[n] \mid \tag{2.9}$$

The drawback of this method is object vanishing. The exponential filter incorporates any foreground element in the background after a certain time. This duration depends on the learning rate α. Mathematically, the step response of exponential filter reaches around 2/3 after the time constant τ. The relationship between the learning rate and the time constant, given the sampling period ΔT, is:

$$\alpha = e^{-\Delta T/\tau} \quad \longrightarrow \quad \tau = -\frac{\Delta T}{log(\alpha)} \tag{2.10}$$

For division implementation in microcontroller, alpha is a power of two ($2^{-7} \approx 0,008$). According to equation 2.10, it achieve around 10 s of time constant, that is a reasonable duration for people detection in outdoor. This parameter can be tuned according to the specific application requirements.

The problem described is presence detection. Formally, it is a binary classification between the two classes "person" and "no person" in the field of view.

Table 2.1 describes the network layers. The other common parameters between layers are kernel size 3 and stride 1. No max-pooling and no padding is applied: the number of pixel is already low (8x8) and decreasing it by pooling would harm the classification. We used no padding, adding extra pixels will change the image significantly. The activation function of each convolutional layer is the Rectified Linear Unit (ReLU). Finally, the last dense layer uses sigmoid as activation function, because a binary classification is required. The sigmoid represents a probability, thus we split the possible outputs in two classes with a threshold of 0.5.

TABLE 2.1: Layer descriptions and bytes needed in an 8 bit format for weights and intermediate outputs

	Layer Type	Filter Shape	Filter Size	Out Shape	Out Size
#0	Input	-	-	8x8x1	64B
#1	Conv.	3x3x1x4	36B	6x6x4	144B
#2	Conv.	3x3x4x8	288B	4x4x8	128B
#3	Conv.	3x3x8x16	1152B	2x2x16	64B
#4	F. C.	16x2x2	64B	1	1B

We used the Python programming language and the framework Tensorflow. The network takes the images after background subtraction as an input. Then, we split the Dataset in training, validation and test in 60%-20%-20%. Cross entropy is the cost function and we

TABLE 2.2: The value of parameters used in the methods, with the corresponding reference to the equation

Equation	Symbol	Value
exponential filter constant	C	0.9
background refresh rate b. correlation (2.3)	α	0.99
window size b. correlation (2.4)	c	2
background refresh rate b. v. detection (2.6)	α	0.999

used the Adam Optimizer to minimize the it, with a starting learning rate of 0.001. We stop the training after 1000 epochs and in each epoch the best model on validation set is saved. Then we used the test set to evaluate the final performance.

2.3.3 Adaptive Threshold

In order to test the algorithms and acquire data we have built a prototype. We have fixed the Grid-EYE camera at 2.85 m of height, inclined 60 degree in order to cover an area of 2m x 1.4m. The data is saved and then processed on a computer, simulating a real-time situation.

We have collected a dataset of 70 independent acquisitions, each one with 4 instances of a person crossing the scene at different distances. Each recording lasts for 16 seconds (160 frames), and for half of the acquisition, the human subject moves with walking speed in the whole field of view of the camera. Each acquisition was collected in a different situation, during a time span of 4 months and with background temperatures ranging from 5 to 30 C°. Labelling is done by means of temporization of crossing during acquisition, tagging the starting and ending instant. It is done manually and for this reason, we added a tolerance of 300 ms. This correction value is extracted from collected data as average of the synchronization errors detected in those acquisition samples where signal to noise ratio is high and therefore it is very easily identified.

The collected data was used to tune the algorithm parameters and to evaluate their performance. In fact, we optimize the neural networks weights, the constant of the exponential filter, the refresh rate in background extimation and the window size for the background correlation technique. The pairs parameter-value obtained are in table 2.2.

With the labelled dataset, we have implemented a script able to find the optimal threshold for each acquisition. We have chosen to define as optimal the threshold that maximizes the F1 score. F1 is calculated with the general equation

$$F1_\beta = (1+\beta^2) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall} \tag{2.11}$$

with $\beta = 1.2$ to give more importance to recall than to precision, as required in our application scene.

Thresholds follow a quadratic trend: they become lower when the background temperature reaches value around 20-25 C°. This is due to the similarity with the human temperature in external environment (with dresses, ecc..). When temperature rises, the human body becomes colder than the background, therefore it can be distinguished very well. Figure 2.5 shows this behaviour for the background correlation case. Similar curves have been extracted in the other three cases. We propose that using the threshold given by the background temperature increases the robustness of the algorithms to different environmental settings. When the background temperature is around 20-25 degrees, a lower threshold should be used to get the small differences between humans and the environment. When the weather is far from 20-25 degrees, it is worthy of raising the threshold because the human temperature is very different and easy to distinguish. Increasing the threshold avoids that small fluctuations of temperature

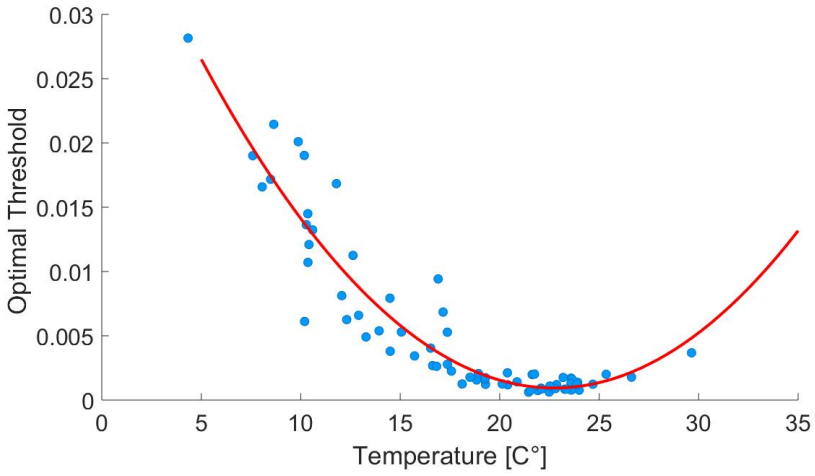

FIGURE 2.5: Optimal thresholds and quadratic interpolation: results follow a quadratic trend

in the environment are detected as a person. Thanks to the interpolated curve it is possible to apply a threshold for each background temperature, measured from the environment.

2.3.4 Embedded Programming for the Convolutional Neural Network

To compute the output of the neural network on a microcontroller, we used the CMSIS-NN libraries. This framework achieves performance optimization of neural networks on resource-constrained microcontroller based platforms [45]. The calculation uses low-precision fixed point representation (e.g. 8 or 16 bit), to optimize the memory footprint and avoid the floating point computation. Moreover, the functions take advantage of the SIMD instructions to exploit parallel computing, always present in convolutional operations.

Many compatibility issue should be managed in the conversion from the computer to the embedded programming. Each weight is quantized to 8-bit fixed point format. To decide how many digits should be used for the integer part, we check the value of all the weights. The values are between -1 to 1, so we use all the bits to describe the digits, and 1 for the sign. In this way, the range of possible number is $[-1, 1-2^{-7}]$ with a resolution of 2^{-7}. The intermediate values (the output of each convolutional layer) is hard to predict as it varies with different input. One possible solution is to try the whole space of input, and determine the range of bits to represent decimal value for each intermediate value and preprocessing output.

This approach is not optimal for our system, because background subtraction output hardly reach the ideal maximum value (an item of 80° appears in a background model of $0\ ^{\circ}$). Thus, we decided to use the training set to estimate the range of the value in practice. We evaluated how many numbers goes above a certain power of two. If the percentage of this numbers is negligible (1%), we use the respective number of digits for the integer, clipping the value above. This is a middle ground between loosing resolution (decimals) and clipping the unlikely integer.

Finally a program export all these information. Firstly, the weights are quantized according to this formula

$$W_q = round(W * 2^n) \tag{2.12}$$

where W_q is the quantized weight, W is the floating point weight in Tensorflow and n is the number of digits used to describe the decimal part. If the output number is bigger than 2^7 or smaller than $-(2^7-1)$, it is clipped to these values.

A program saves the weights in a header file, in a flattened way ([out channel, filter size, filter size,in channel]) according to the one expected from the CMSIS functions. The script automatically saves the number of integer digits for intermediates output in a header files, together with others hyperparameters like number of filters, output-image dimensions, padding and kernel size.

2.3.5 Hardware Platform

In order to evaluate the power consumption and the real-time performance of the neural network implementation, we moved from an offline PC-based system to a generic low power hardware.

The board is the STM NUCLEO-L476RG. The main feature of its microcontroller is a Cortex-M4 core, up to 80 MHz clock and the possibility to reach very low consumption (down to 8 µA) in stop modes. The sensor outputs data every 100 ms. The forward propagation in the neural network takes less than 5% of this time in maximum frequency mode. As a result, the microcontroller has an idle period and it lets the system go in low power mode to reduce the overall power consumption. To wake up the system every 100 ms, the microcontroller has an integrated low power timer that enable periodic interrupt also during the sleeping mode, with a low-frequency clock of 32 kHz. Then when the system is woken up, the clock is set to 80 MHz and acquisition restarts.

2.4 Results

To evaluate the system quality we have used the optimal threshold and we have compared the results with ground truth. The evaluation criteria are defined in this way:

- **False Positive:** the output goes above the threshold at least once when nobody is in the field of view
- **True Positive:** the output goes above the threshold at least once when there is a person in the field of view
- **False Negative:** the output never goes above the threshold during all the crossing
- **True Negative:** the output remains below the threshold when nobody is in the field of view

These definitions come from the application scenario. On the other hand, they are the same in most of the system related to people detection.

Figure 2.6 shows the F1 score for different kinds of thresholds. The yellow bars are referred to a fixed threshold for all the dataset. Red bars are relative to results achieved by using interpolated thresholds from quadratic curves (figure 2.4). Graphs show that F1 score using interpolated thresholds is clearly higher than the one using a fixed threshold.

Background correlation is the most sensitive to threshold. In fact, with this technique performance with fixed threshold is very poor. The five techniques have similar performance in terms of F1 score, and the Convolutional Neural Networks is the best performing. For this reason, in the next section we focus on its implementation in an embedded platform.

Results in general demonstrate that usage of thermopile arrays for people detection is possible in outdoor conditions. In addition, the adaptive thresholds described in this work achieve the best results.

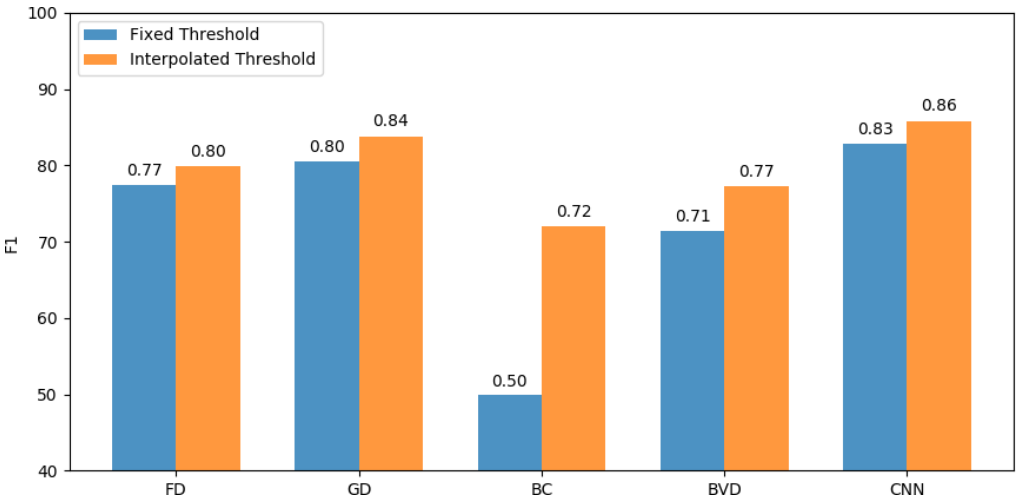

FIGURE 2.6: Results for the same data but with different thresholds

2.4.1 Hardware Implementation Results

In this section, we describe the hardware performance in terms of execution time, power consumption and memory footprint for the best performing algorithm, namely the neural network. Then, a different firmware enables the simulation of test images on the microcontroller and let us calculate the loss in classification performance due to the quantization process.

TABLE 2.3: Overall Results

Power Consumption [mW]	16.5
Power Consumption μc [mW]	2.3
Execution Time [ms]	4.01
Text [B]	19688
BSS [B]	2712
Data [B]	2680

Power Consumption: As explained in section 2.3.5, the algorithm goes in low power when it processes the image and it wakes up when the image from the sensor is ready. So the current absorbed is considerable during acquisition and processing, but it is negligible during stop mode. The device used for current measuring is the Real-Time Current Monitor EE203. STM board let the costumers to measure the consumption of microcontroller, without considering the current flowing in regulators, debugger and other devices on the board. Anyhow, the sensor is part of the system itself, so we have measured current absorbed from the voltage supply, with and without the sensor connected.

Computation Time: Part of the time is needed to load the image from the sensor via I2C bus, then it computes the background subtraction and lastly it performs the classification. Therefore, computation time is the time elapsed between starting reading and classification output. Total time is 4.01 ms; figure 2.4.1 shows the different parts in the active phase. Reading the data takes most of the time, background subtraction is negligible and classification takes around 2 ms.

Memory Footprint The compiler returns Text, BSS and Data values for the implemented firmware. Biggest size is text, that fits into around 20 kB Flash. The other two values in table 2.3, BSS and Data, are respectively initialized and uninitialized variables. The sum of the two shows a total of around 6 kB in the RAM.

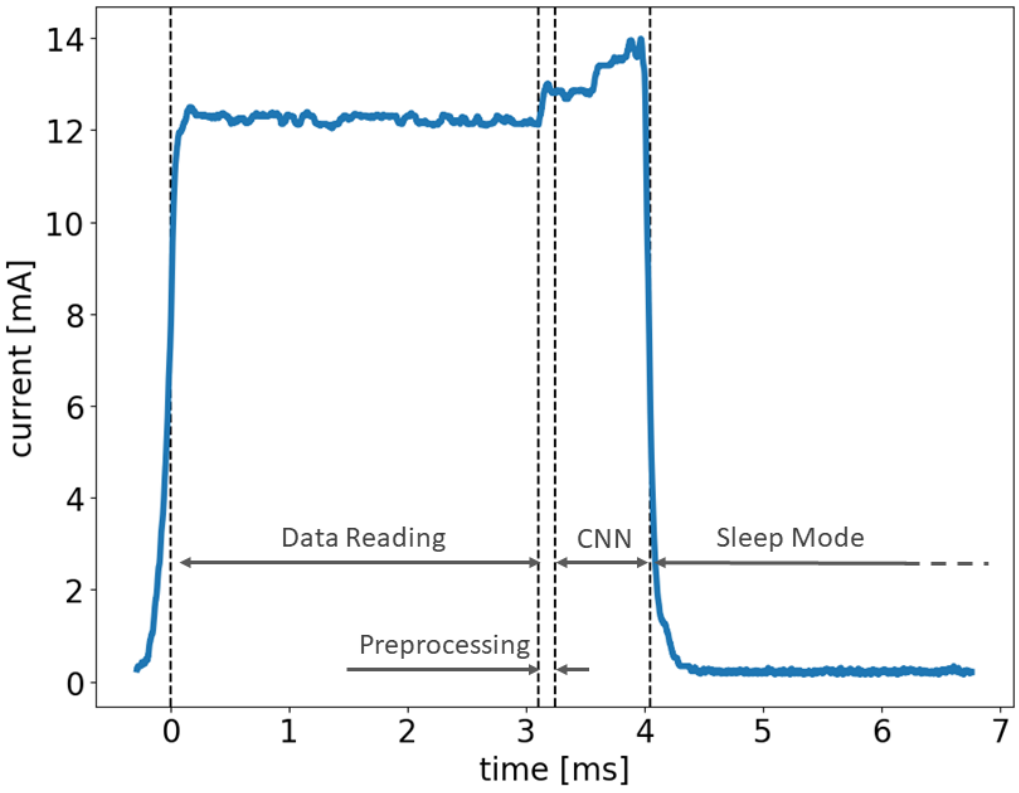

FIGURE 2.7: Current consumption during the active phase.

Classification Performance To evaluate the differences, we computed the accuracy of the test Dataset with Tensoflow on the PC. Then we send the Dataset via UART to the board, already preprocessed on the computer. This is to focus on the loss due to the classification, by not considering the contribution of preprocessing. We sent all the three sets (training, validation and test) to have a more generic description of the losses.

TABLE 2.4: Loss in classification between 32 bit floating point and 8 bit fixed point representation

Accuracy	Tensorflow [%]	CMSIS-NN [%]	Loss[%]
Train	81.1	80.9	0.2
Validation	77.4	76.4	1.0
Test	76.9	76.7	0.2

2.5 Conclusions

Summing up, we demonstrated that, in the case of the Convolutional Neural Network, low precision format (8 bit fixed-point) does not affect significantly the performance, and it makes the system lose at maximum 1% of accuracy. Finally, it lets the network fits in just ~20 kB of Flash and ~6 KB of RAM.

Overall, the case study presented here was a first exploratory activity to apply the main elements of our approach: bringing intelligence at the "thing" level keeping an eye on the balance between performance and energy efficiency, acting at both hardware and software level. The underneath application theme is to enable functionalities at the edge of an IoT infrastructure for context extraction in outdoor scenarios. This is because we want to apply

our research to smart cities. In the next chapter, we will move to a different sensor, but we will dig further in the exploration of deep learning techniques to make the "things" even more smarter. We thus switch the application domain to Audio Event Detection, where the algorithm is still a convolutional neural network, and the quantization procedure in this chapter is the starting point for a more advanced approach.

Chapter 3

Neural Network Distillation Targeting Embedded Platforms

3.1 Overview

The work on the low-resolution thermal sensor shows the predominance of neural networks against classic computer vision approaches. At the same time, the images coming from the sensor do not carry enough details to identify different events or extract context information. To extract richer content and perform more advanced classification tasks a different sensor will be required, e.g., a high-resolution vision sensor. Examples of applications enabled by a camera could be trackers/counters of cars, people, bicycles. A well-known drawback, however, is their power consumption. Moreover, the use of cameras in public places raises privacy concerns.

Audio sensors, i.e. microphones, are more energy-efficient components than vision ones for IoT end nodes. They likewise enable context information extraction, keeping the consumption of the sensor itself under mW values. Everyday experience tells us that audio signals provide useful and different information, such as understanding a scene we are in or observing or identifying specific events. As in the study done with the thermal sensor, the architecture for audio analysis is based on convolutional networks, this time on spectrograms. It means that most of the explorations for the embedded implementation of the people detection system described in the previous section is reused and improved in this chapter. While in the case of the thermal sensor the model itself was shallow, and the main limitation is the relatively low information in the input data, here the problem is on the computational effort. State of the art sound processing techniques for classification tasks, such as audio event detection or scene recognition, are not tailored for embedded system. In other words, the number of operations required and the amount of RAM needed exceeds microcontroller capabilities. However, low-power microcontrollers are, in many cases, the only possible choice to implement an energy independent end node and thus a pervasive IoT application.

Our proposed approach differentiates itself from those available in literature in many directions since it attempts to pull together the benefits of the multiple compression methods described in the introduction (see paragraph 1.4.1). We start from a large model and compress it. However, instead of just focusing on pruning or weight sharing, which provides limited memory reduction without decreasing the processing time, we design a minimal target network and use KD to train it (instead of training from scratch as in [79, 92, 105]). Note that distillation is typically employed to obtain limited network reductions, while here, we target extremely high compression factors. On top of this heavy compression, we apply a stochastic weight and buffer quantization without the need for retraining the network. To confirm all the estimation in a real-life scenario, we implemented the student network on a microcontroller with resources typical of an IoT end-node, and we evaluated the accuracy of the actual implementation. The final result definitely respects an IoT end-node constraint, using just 34.3 kB of RAM and 5.5 mW of power consumption.

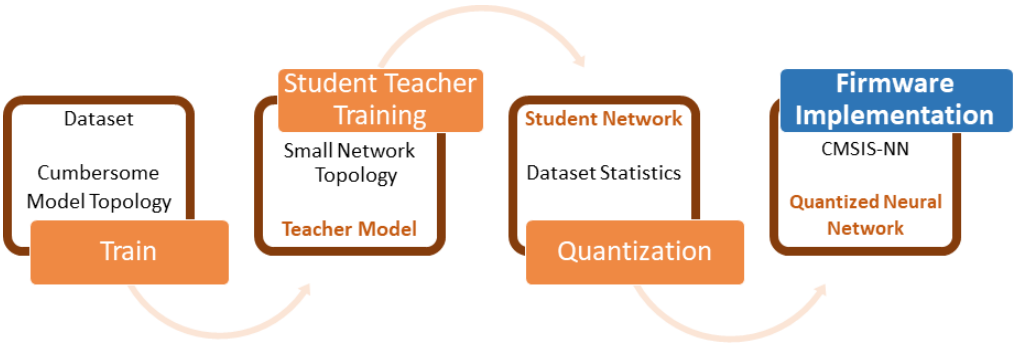

FIGURE 3.1: From state-of-the-art to IoT: Starting from a sound event detection dataset and a state-of-the-art network topology that performs well on the problem, we apply the student-teacher approach to train a much smaller network that would fit on IoT devices. Finally, we quantize the compact network and write the firmware for low-energy platforms.

3.2 Knowledge Distillation

In this section, we give a brief overview of the Student-Teacher approach.

Considering a generic neural architecture where a feature extractor is followed by classifier, several strategies can be adopted to train the student network. Figure 3.2 graphically shows the distillation process, where the upper part represents the teacher network and the lower part is the student.

The first training strategy uses the hard (one-hot) labels, which corresponds to training the network from scratch without any other knowledge. Since we are dealing with a multi-class classification problem, the loss is based on cross-entropy:

$$\mathcal{L}_{\mathrm{h}}(\mathbf{X}) = -\sum_{n=1}^{N}\sum_{c=1}^{C}\mathbf{y}_n \log(p_c(\mathbf{x}_n)), \tag{3.1}$$

where $\mathbf{X} = \{\mathbf{x}_1, \dots \mathbf{x}_N\}$ is the set of the network input features, N is the number of samples, C is the number of classes, $\mathbf{Y} = \{\mathbf{y}_1, \dots \mathbf{y}_N\}$ are the one-hot labels, $\mathbf{y}_n$ is a C-dimensional binary vector and $p_c(\mathbf{x}_n)$ is the network output for class $c = \{1, \dots, C\}$ given the input vector $\mathbf{x}_n$. Denoting as $\overline{p_c}(\mathbf{x}_n)$ the output of the teacher network, the soft loss $\mathcal{L}_{\mathrm{s}}(\mathbf{X})$ can be formulated by replacing the hard labels with the teacher output:

$$\mathcal{L}_{\mathrm{s}}(\mathbf{X}) = -\sum_{n=1}^{N}\sum_{c=1}^{C}\overline{p_c}(\mathbf{x}_n) \log(p_c(\mathbf{x}_n)). \tag{3.2}$$

Combining equations 3.1 and 3.2 leads to the approach in [35]:

$$\mathcal{L}_{\mathrm{h,s}}(\mathbf{X}) = \alpha_h \mathcal{L}_{\mathrm{h}}(\mathbf{X}) + \alpha_s \mathcal{L}_{\mathrm{s}}(\mathbf{X}), \tag{3.3}$$

where α_h and α_s are combination weights. A further strategy, in line with [76], is to make the student learn how to replicate the features produced by the teacher, leaving the classification independent. The embedding loss $\mathcal{L}_{\mathrm{e}}(\mathbf{X})$ is thus defined as:

$$\mathcal{L}_{\mathrm{e}}(\mathbf{X}) = \sum_{n=1}^{N} ||v_t(\mathbf{x}_n) - v_s(\mathbf{x}_n)||^2, \tag{3.4}$$

where $v_t(\cdot)$ and $v_s(\cdot)$ are the feature vectors produced by the teacher and student networks respectively. Finally, we consider a compound loss obtained as linear combination of the

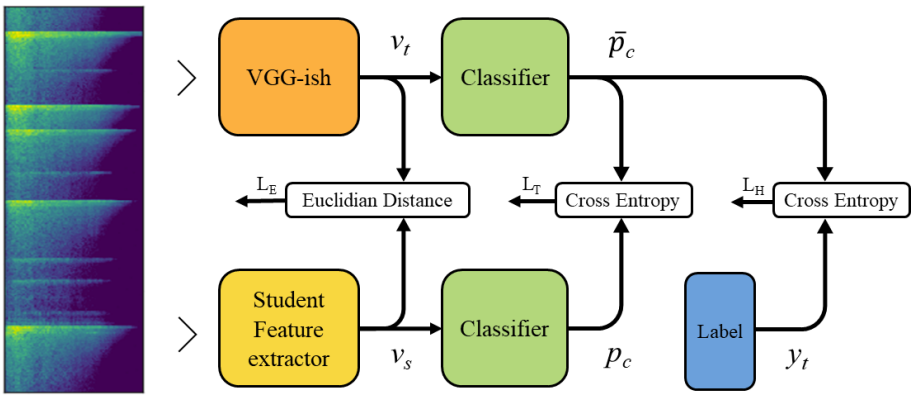

FIGURE 3.2: Block diagram of KD using teacher intermediate features (L_E), soft teacher output (L_T) and dataset labels (L_H)

three losses introduced above:

$$\mathcal{L}(\mathbf{X}) = \alpha_h \mathcal{L}_{\mathrm{h}}(\mathbf{X}) + \alpha_s \mathcal{L}_{\mathrm{s}}(\mathbf{X}) + \alpha_e \mathcal{L}_{\mathrm{e}}(\mathbf{X}) \tag{3.5}$$

3.2.1 Dataset

For SED in outdoor urban environments, three datasets are often used in literature: *UrbanSound8K* [81], AudioSet [22], ESC50 [71] and TUT Sound events 2017 [56]. The latter is particularly attractive because it features real recordings and several comparative methods are available thanks to the related DCASE challenges [57]. Unfortunately, the task is very hard and the state-of-the-art accuracy is rather low. Moreover, the class distribution is highly unbalanced towards one class. Therefore the dataset does not allow a fair evaluation of KD methods. ESC50 is also rather in line with our application scenario, but its size is relatively small (3 minutes of audio per class) and does not allow generalizing the results. Finally, we also discarded AudioSet because its video-based labels, referring to scenes instead of isolated sound-events, require consistent additional work to be aligned with the label required for our analysis.

Therefore, we focused on *UrbanSound8K*. It includes 8732 audio samples related to the city environment, with different sampling rates, number of channels and a maximum length of 4 seconds. Each recording has a unique label among 10 possible classes: air conditioner, car horn, children playing, dog bark, drilling, engine idling, gun shot, jackhammer, siren, and street music. All clips are taken from Freesound[1], a vast collaborative database of audio samples. Following the recipe reported in [81], we use 10-fold cross-validation and average scores. However, we took one additional fold for validation: 8 folds are used as training data, one is used as validation and the remaining one for test (training-validation-test ratio is 0.8-0.1-0.1). Validation fold is one index less than the test one (for example, when the test fold is 9, the validation fold is 8). Performance is measured in terms of classification accuracy. In all experiments, the dataset is augmented through pitch shifting [80], with both positive (up) and negative (down) semitones with values -2,-1,1,2.

3.2.2 Teacher

State-of-the-art solutions for SED are mostly CNN fed with mel-spectrogram [70][106][104]. However, to fully exploit the potential of the KD approach, rather than training from scratch

[1] "http://www.freesound.org"

TABLE 3.1: Architecture of the models under analysis. "-" means that the layer is not active, "x" means that the layer is active.

Layer	VGGish/M_{70M}	M_{20M}	M_{2M}	M_{200k}	M_{20k}
Conv1	64	64	32	8	4
Pool1	x	x	x	x	x
Conv2	128	128	64	16	8
Pool2	x	x	x	x	x
Conv3	256	256	128	32	16
Conv4	256	-	-	-	-
Pool3	x	x	x	x	x
Conv5	512	256	128	64	16
Conv6	512	-	-	-	-
Pool4	x	x	x	x	x
Conv7	-	-	-	64	32
Pool5	-	-	-	x	x
FC1	4096	2048	512	256	64
FC2	4096	2048	-	-	-
FC3	128	128	128	128	128
BatchNorm	x	x	x	x	x
GRU	20	20	20	20	20
FC4	10	10	10	10	10
#Param	~72.1M	~18.0M	~1.88M	~202k	~30.6k

our big CNN, we employed the publicly available VGGish[2] feature extractor [33], followed by a classification stage tailored on *UrbanSound8K*. VGGish is trained on the Youtube-8M Dataset [23] and it is expected to generalize well to other application contexts. Note that this fact introduces a further novelty in our work since distillation is performed on a different dataset than that used in the original training. VGGish converts 960 ms audio input mel spectrogram into a 128 dimensional embedding that can be used as input for a further classification model. The classifier can be shallow as the VGGish embeddings are more semantically representative than raw audio features. The VGGish architecture is described in Table 3.1. For all the convolutional layers the kernel size is 3, the stride is 1 and the activation function is ReLu. Max pooling layers are implemented with a 2x2 kernel and a stride of 2.

The classifier consists of a gated recurrent unit (GRU) followed by a fully connected layers and maps the VGGish embeddings into the 10 classes of *UrbanSound8K*. A BatchNormalization layer is inserted between the feature extractor and the classifier to accelerate training.

VGGish expects as input 960 ms of audio signal sampled at 16 kHz. Each clip in *UrbanSound8K* is resampled and padded or cropped to get a length of 960*4=3840 ms. The resulting signal is divided into 4 non-overlapping 960 ms frames. For each frame, the short-time Fourier transform is computed on 25 ms windows every 10 ms. The resulting spectrogram is integrated into 64 mel-spaced frequency bins, covering the range 125-7500 Hz, and log-transformed. This gives 4 patches of 96 x 64 bins that form the input of the VGGish.

[2] "https://github.com/tensorflow/models/tree/master/research/audioset"

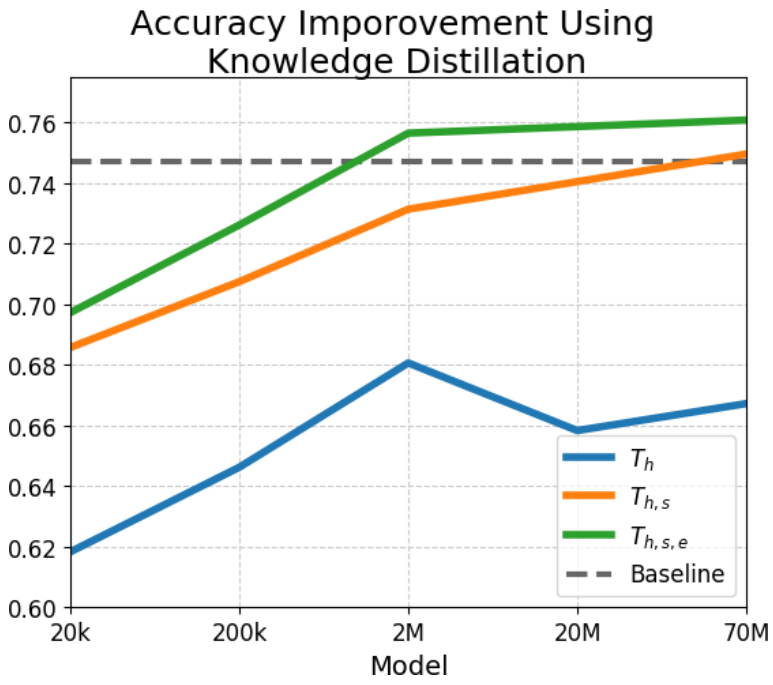

FIGURE 3.3: Average accuracy for the 4 different models in table 3.1 plus model M_{70M}. Each line represents a different training strategy. T_h is standard training, using just hard labels. $T_{h,s}$ uses both hard and soft labels. $T_{h,s,e}$ minimize also the feature vector produced by an intermediate level. Dashed line is the teacher accuracy, here referred as baseline.

3.2.3 Distillation

In this section, we analyze how the compression factor impacts on the final classification accuracy, using the standard loss and the loss compound described above. Although our goal is to fit the classifier on an IoT device, we consider four different degrees of compression, to better assess the potential and limits of the proposed approach, as reported in Table 3.1. An heuristic adjustment of the layers and the number of filters is used to design the reduced networks. The model subscript M_x approximately represents the number of parameters of the upstream part. Note that the student networks drastically reduce the feature extractor only. The classifier is re-trained, but it keeps the same architecture, i.e. a 20-unit recurrent layer followed by a 10-unit fully connected layers.

Figure 3.3 reports the results in terms of classification accuracy for the 4 models and when training from scratch on *UrbanSound8K* and when distilling from the VGGish teacher. In addition, we consider the model M_{70M}, which is a replica of the VGGish's architecture. All models considerably improve thanks to knowledge distillation with respect to being trained from scratch. Note also how the large models (M_{20M}, M_{70M}) perform much worse than VGGish and slightly worse than M_{2M} when trained from scratch: this is due to the fact that *UrbanSound8K* is not large enough for such huge networks. Finally, it is worth noting that the VGGish baseline is outperformed also by all models with more than 2M parameters. This is mostly due to the domain adaptation that we are implicitly applying when distilling knowledge from the teacher. As a matter of fact, the student feature extractor is tailored to the new in-domain data from *UrbanSound8K*, providing an improvement over the more general purpose VGGish feature extractor trained on Youtube8-M.

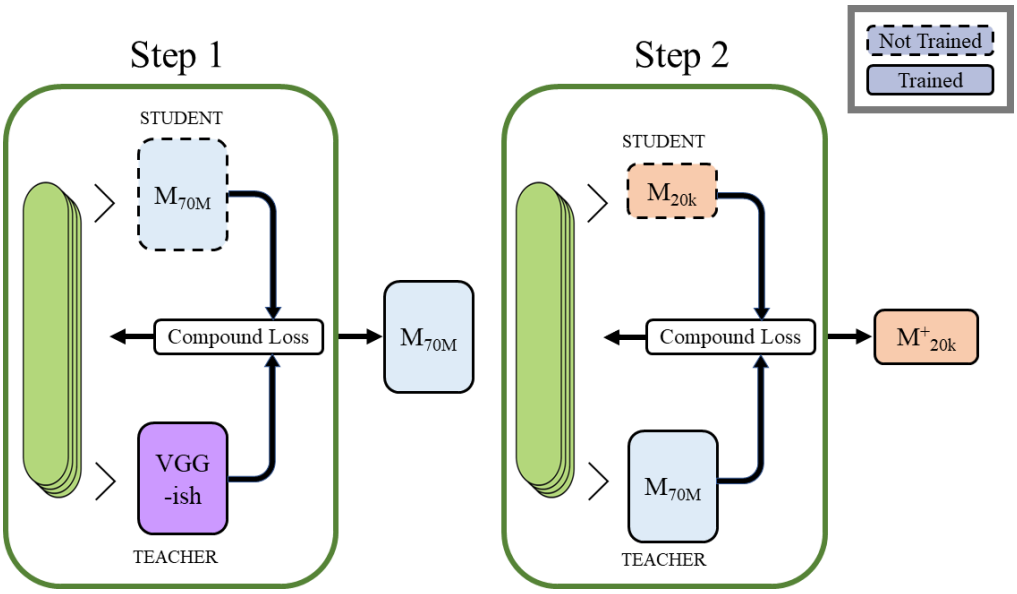

FIGURE 3.4: Two stage distillation: M_{70M} is firstly trained from VGGish and the student M_{20k} is distilled from M_{70M}

3.2.4 Two-Stage Distillation

The previous analysis, showing that we are jointly performing domain adaptation and parameter reduction, suggested us to investigate whether a further improvement can be achieved by separating these two processes. Figure 3.4 describes our proposed 2-stage distillation, where we train the smallest network using the M_{70M} network as teacher. Note that M_{70M} is still trained using the compound loss in Eq. 3.5. Results are reported in Table 3.2 in comparison against training from scratch and using distillation from VGGish. The proposed approach achieves a 72.67% accuracy on the test set, providing a 3 points improvement over a more traditional distillation strategy (69.7%). We expect an increase in performance in the small network because the new teacher is performing better on the dataset (%76.07 versus %74.7 of the baseline). Interestingly, the new M_{20k} model is just 2 points below the VGGish baseline and 4 points below M_{70M}, in spite of using less than 0.0424% of the parameter and 0.12% of the operation.

TABLE 3.2: Accuracy of the M_{20k} for: training from scratch, distillation from VGG-ish and distillation from M_{70M}

	Standard Training	From VGG-ish	From M_{70M}
M_{20k}	61.83	69.72	72.67

3.3 Hardware Resources and Network Requirements

3.3.1 Approximate Hardware Requirements per Model

One interesting aspect towards tayloring our distilled networks for a resource constrained platform is to understand what are the computational and hardware requirements for a given model, or to understand what accuracy would be achievable with a given platform.

TABLE 3.3: Approximate computational and memory requirements for each network

	VGGish	M_{20M}	M_{2M}	M_{200k}	M_{20k}
#Param	~72.1M	~18.0M	~1.88M	~202k	~30.6k
#Operations	~1.72G	~608M	~148M	~13.6M	~2.11M
#Buffer [B]	~614k	~602k	~301k	~76.0k	~34.3k

Table 3.3 reports an approximation of the computational requirements of each network: number of parameters to store, number of operations and buffer sizes. For memory requirements, we refer to an implementation using 8-bit quantized weights and buffers as specified in the CMSIS-NN library.

With an 8-bit quantization, each parameter takes one byte; therefore, the **non-volatile memory** in bytes needed by a model equals the total number of weights and biases (first row of Table 3.3).

Buffers keep the outputs of each layer available during propagation and are stored in the **RAM**. The size of these buffers depends on the output dimensionality of each layer. However, the total amount of run-time memory required depends heavily on how efficiently buffers are implemented. The third row of Table 3.3 reports the RAM requirements of each model given the buffer design described in section X.

The **number of operations** (both multiplications and sums) depends on the layer type: in convolutional and maxpooling layers, each output pixel comes from a filter application. Given kernel size k and number of input channels c, each filter requires the following operations:

$$Ops_{filter} = 2 \cdot c \cdot k^2. \tag{3.6}$$

Therefore, the number of operations for each convolutional and maxpooling layer is:

$$Ops_{Conv} = Ops_{filter} \cdot out_H \cdot out_W \cdot out_C, \tag{3.7}$$

where out_H, out_W, andout_C are the output height, width an channel respectively.

For dense and GRU layers the number of operations is the number of matrix multiplications:

$$Ops_{DotProduct} = 2 \cdot in_{shape} \cdot out_{shape}. \tag{3.8}$$

Gated recurrent unit requires also three element-wise products.

3.3.2 Selection of the Device Class

Table 3.4 shows a non-exhaustive list of processing platforms potentially adequate to be integrated into an end-device, with their power consumption and memory. In this section, we provide a qualitative analysis of what devices would be able to run the distilled models presented in the previous sections. This analysis is inevitably rough as many figures are approximated and the actual values cannot be derived analytically. Additionally, we are not considering the time and resources required for other processing stages (e.g., Mel-bins extraction); thus, the constraints are rather relaxed with respect to the actual application. Nevertheless, the devices in Table 3.4 show substantial differences in MIPS and memory (in the order of powers of 10); therefore, the results of this study are still valid as long as we refer to classes of devices.

Computational Cost. To ensure real time classification, the network must process each 960 ms audio frame (~1 second) before the next frame arrives. Thus, classification time must be shorter than 1 second. Unfortunately, converting exactly the Million Operation Per Second (MOPS) required by each model in the MIPS available on a given device, as reported in

TABLE 3.4: Examples of embedded platforms and their hardware capabilities.

Board Name	Flash[KB]	RAM[KB]	Power [mW]	MIPS
Arduino	32	2	60	20
ChipKit uc32	512	32	181	124.8
STM32L476RG	1024	128	26	80
TI MSP432P4111	2048	256	23	58.56
BeagleBone Black	Ext	524288	2300	1607
Raspberry Pi 3 B+	Ext	1048576	5500	2800

the datasheet, is not feasible. As a matter of facts, the number of instructions required by an operation depends on many factors, the most important being the actual implementation. In this analysis, we rely on the assumption that, on average, the number of operations is equal to the number of instructions. Typically, instructions are more than operations because each operation involves branch, load and store instructions. However, most of 32-bit microcontrollers support single instruction multiple data (SIMD), which allows up to four instructions in one clock cycle. We assume that these two effects are balanced and the number of operations is equal to the number of instructions. In the next sections, we will demonstrate that the actual throughput (operation per instruction) is larger than 1, but it gets close enough (1.8) in some situations. Given this assumption, in combination with the fact that other processing stages are not being accounted for, the MIPS available on the device must be larger (with some margin) than the MOPS needed in one forward propagation.

Memory constraints are also relevant: the RAM on-board must contain the intermediate values and the non-volatile memory should contain all network parameters. However, non-volatile memory is not the main limit in our examples: whenever the network does not fit in the flash memory it does not respect one of the others two parameters (RAM or MIPS).

Figure 3.5 roughly compares the devices in Table 3.4 in terms of RAM and MIPS limitations against the compressed models each device can afford. None of the proposed models

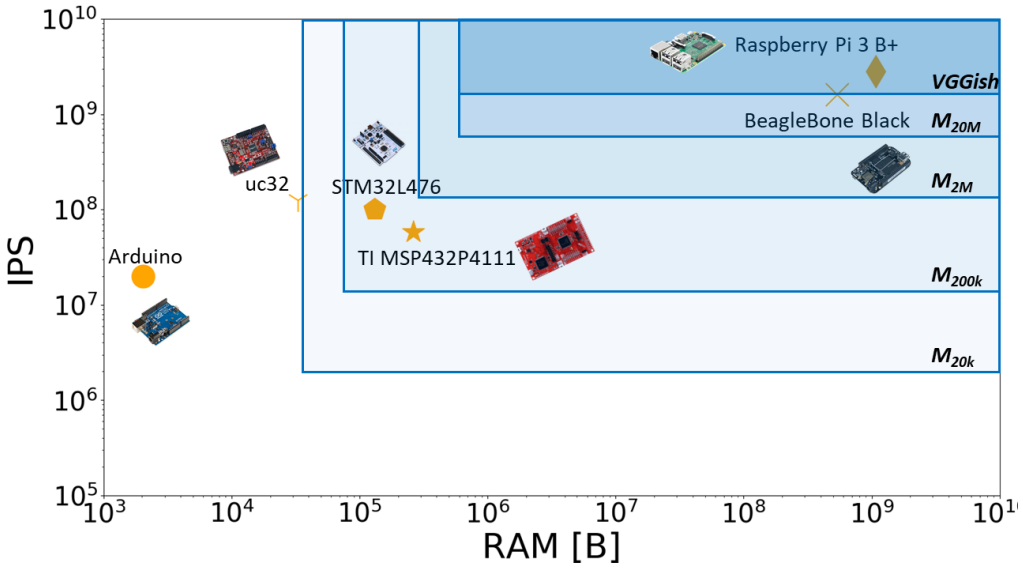

FIGURE 3.5: Models requirements vs hardware capabilities: a qualitative analysis Rectangles define network requirements : in terms of MIPS and RAM.

fits in the Arduino platform due to its limited RAM. The last two platforms of Table 3.4 (BeagleBone Black and Raspberry Pi 3 B+) have enough RAM and MIPS to handle all models

(actually VGGish fits only in Raspberrypi 3 B+). However, their power consumption is in the range of Watts that is not suitable for IoT applications. Therefore only devices ChipKit uc32, STM32L476RG, and TI MSP432P4111, which are approximately in the same class, are potentially suitable to host the distilled models. Uc32 is faster than both STM32 and TI platforms and would comfortably run the two smallest models in real-time, but its 32KB RAM is not large enough to fit even the ~34.3 kB needed for the buffers of the smallest model M_{20k}. Concluding, M_{20k} and M_{200k} are the only suitable models for the target application and can be implemented in both the STM32L476RG and TI MSP432P4111 platforms. In the next section, we detail our implementation of M_{20k} in the STM32L476RG platform.

3.4 Embedded Programming

For the implementation of our SED system, we selected a Cortex M4 architecture operating up to 80 MHz, which is a good compromise between efficiency in processing and flexibility in power management. In particular, we worked on an STM3276RG Nucleo board as a development platform. Thus, the reference values for current consumption are 420 nA in standby mode and 100 µA/MHz in full running mode. The Floating Point Unit (FPU) features single precision and implements a full set of optimized Digital Signal Processing (DSP) instructions for fixed-point operations.

Cortex-M4 provides SIMD instructions that operate on 8 or 16-bit integers. They are powerful for processing data such as video or audio, when full 32-bit precision is not necessarily required. The SIMD instructions allow 2 x 16 bit or 4 x 8 bit operations to be performed in parallel [52].

3.4.1 Quantization

Feed-forward propagation through a neural network requires vector/matrix/tensor multiplication and convolution. Therefore, all the core features employed in signal processing can be used for neural network computations. For this reason, ARM developed the CMSIS-NN framework for neural network propagation on top of DSP libraries [45]. The CMSIS-NN library maximizes performance and energy efficiency of common deep learning kernels on top of Cortex-M series cores.

Like for DSP, truncation of floating point numbers to 8 or 16-bit fixed point numbers improves the execution time and reduces the memory footprint. According to [45], 8-bit quantization achieves 4.6X improvement in runtime/throughput and 4.9X improvement in energy efficiency in image classification with CIFAR10 dataset. On the other hand, since quantization implies loss of precision, we could expect a direct impact on the final prediction performance. However, the authors of [50] experimented different kinds of quantization in image classification, achieving a 20% drop in model size without significant loss of accuracy. In the following, we describe the quantization process from a 32-bit floating-point to an 8-bit fixed-point representation. From now on, we will assume that floating point numbers has infinite precision and we will use the nomenclature typically used for DSP.

Quantization describes a real number with finite resolution. When a uniform quantization is applied, three parameters are used to define the fixed point representation: bit-width, step-size (resolution) and dynamic range [50]. These parameters are correlated by the following expression:

$$Range = Stepsize \cdot 2^{bitwidth-1}, \tag{3.9}$$

where $bitwidth - 1$ accounts for the bit used to represent the sign. $Stepsize$ is the minimum step between two fixed point numbers and will be always chosen as a power of two for convenience with binary representation.

An equivalent formulation of Eq. 3.9 can be obtained by considering the number of bits used for the decimal and integer parts of numbers:

$$\begin{aligned} Range &= 2^{integer} \\ Stepsize &= 2^{-decimal} \\ bitwidth &= integer + decimal + 1 \end{aligned}$$

where, *integer* is the number of bits used to represent the integer and *decimal* is the number of bits used for the decimal part. +1 in the last equation accounts for the sign bit. Basically, given a fixed number *bitwidth* of bits, increasing the range by increasing the number of bits for the integer part degrades the resolution. Conversely, decreasing *Stepsize*, by allocating more bits to the decimal part, leads to a range reduction.

The quantization error e_q is the difference between the infinite precision number and the quantized representation:

$$e_q = x - Q(x) \tag{3.10}$$

where x and $Q(x)$ are the input and the quantized output. If we treat the input as a random variable with a probability density function f_x (see [84]), the mean square quantization error is the combination of two different errors, namely granular error ($\text{MSE}_{q,g}$) and overload error ($\text{MSE}_{q,o}$)):

$$\begin{aligned} \text{MSE}_{q,g} &= \sum_{i=1}^{\text{N}} \int_{d_i}^{d_{i+1}} (x - Q(x))^2 f_x(x) \text{d}x \\ \text{MSE}_{q,o} &= \int_{-\infty}^{d_0} (x - Q(x))^2 f_x(x) \text{d}x + \\ &+ \int_{-\infty}^{d_N} (x - Q(x))^2 f_x(x) \text{d}x \end{aligned}$$

where N is the number of quantization levels (in our case $2^{(bitwidth)}$) and d_i are the decision levels, i.e. any number between d_i and d_{i-1} is coded with the same fixed-point representation, usually $(d_i + d_{i-1})/2$. The two mean square errors are related to *Stepsize* and *Range* respectively. If *Stepsize* is reduced then $\text{MSE}_{q,g}$ decreases; on the other hand if *Range* is made wider, $\text{MSE}_{q,o}$ decreases.

The figure of merit linked with MSE is the SQNR, defined as:

$$\text{SQNR} = \frac{E\left[x^2\right]}{E\left[e_q^2\right]} = \frac{E\left[x^2\right]}{\text{MSE}_q} \tag{3.11}$$

Therefore, the goal is to define the optimum trade-off between *Range* and *Stepsize* to minimize the SQNR in the target application. In the following, we describe two different strategies to perform quantization using the quantization parameters and its figure of merit described above.

3.4.2 Quantization Design

Applying quantization to neural networks has different requirements with respect to other contexts, as image quantization or audio quantization. In particular, we are not interested in preserving an accurate representation of all activation outputs or weights for each layer. Conversely, we want that the final prediction is as close as possible to the prediction of the network with a 32-bit floating-point representation. To summarize, accuracy is the most relevant metric.

An exhaustive search of all the possible *integer/decimal* combinations, evaluating the final accuracy, is not feasible in practice. In the smallest network, i.e. M_{20k}, the number of *integer/decimal* digits would be 28, and each of them can vary between 0 and 7 (setting the decimal digit decides the integer part when *bitwidth* is fixed). Therefore, the number of possible combinations is given by the permutation with repetition: $7^{28} = 4.5 * 10^{23}$.

The two solutions presented in this paragraph target the maximization of the SQNR or the reduction of the overload error.

The first approach to compute SQNR applies quantization on each variable (weights, activation) one at a time. For the weights, we choose the number of decimals that maximizes the SQNR. For the intermediate outputs we compute the SQNR running a forward propagation in floating-point on the training dataset and testing all the possible integer/decimal values (in the range from 0 to *bitwidth*). Each activation output is analyzed independently, eventually leading to different quantization trade-offs.

This approach relies on [50], where the overall SQNR, here defined as γ, is the harmonic mean of the SQNR of all preceding quantization steps:

$$\frac{1}{\gamma_{\text{output}}} = \frac{1}{\gamma_{a^{(0)}}} + \frac{1}{\gamma_{w^{(1)}}} + \frac{1}{\gamma_{a^{(1)}}} + \cdots + \frac{1}{\gamma_{w^{(L)}}} + \frac{1}{\gamma_{a^{(L)}}} \tag{3.12}$$

It turns out that maximizing the single SQNRs maximizes the overall SQNR, regardless of where quantization happens (at the first layers or at the last layers).

The second approach is based on the different effects of granular and overload errors. It is reasonable that a single number with a high quantization error due to overload will affect the overall forwarding of the neural network. Conversely, the ensemble of small granular quantization error may not change the argmax of softmax layer, that is the value used to determine prediction and accuracy. Therefore, we select the integer/decimal ratio that reduces the probability of having values out of the quantization range. In particular, we set the number of bits for the integer part i such as

$$\min(i) : P(|x| < 2^i) < P_{th} \tag{3.13}$$

where x is the floating-point input and P_{th} is the probability of overload that one is willing to accept. This approach differs from the previous one for three reasons. (i) Numbers with large overload error have *bitwidth* of the same importance than numbers with small overload error. (ii) Thus, each overload is considered uniformly, without accounting for the granular error but taking into only the overload error. (iii) This second approach has a hyper-parameter, i.e. the probability threshold, which requires a fine tuning on the training set. We heuristically set the threshold to 10^{-4}.

We estimate the probability density function $f(x)$ using the whole training set. The percentage of values out of the range is counted for all the possible decimal values involved in the network feed-forward propagation.

The upper part of Figure 3.5 is the histogram of the absolute value of the activation outputs between two consecutive powers of two. Most of the values (more than 10^6) are between -1 and 1 (or equivalently $|x| < 2^0$). Oppositely, just a small set of numbers (around 10^2) are such that $2^2 < |x| < 2^3$; they are depicted in the fourth bar. The central part of Figure 3.6 depicts the relative distrbution for each class of values, e.g. the estimated probability that a value is inside a given range related to two consecutive powers of 2. The values confirm what bars show: values in $|x| < 2^0$ are the most likely and values in $2^2 < |x| < 2^3$ appears only with a probability of 10^{-5}. Finally, the plot at the bottom depicts the SQNR using the number of bits for the decimal part available for a given range in the x axis (the range determines the bits for the integer part). For example, the most left point is the SQNR using 0 bits for integer part (which allows describing numbers in the range

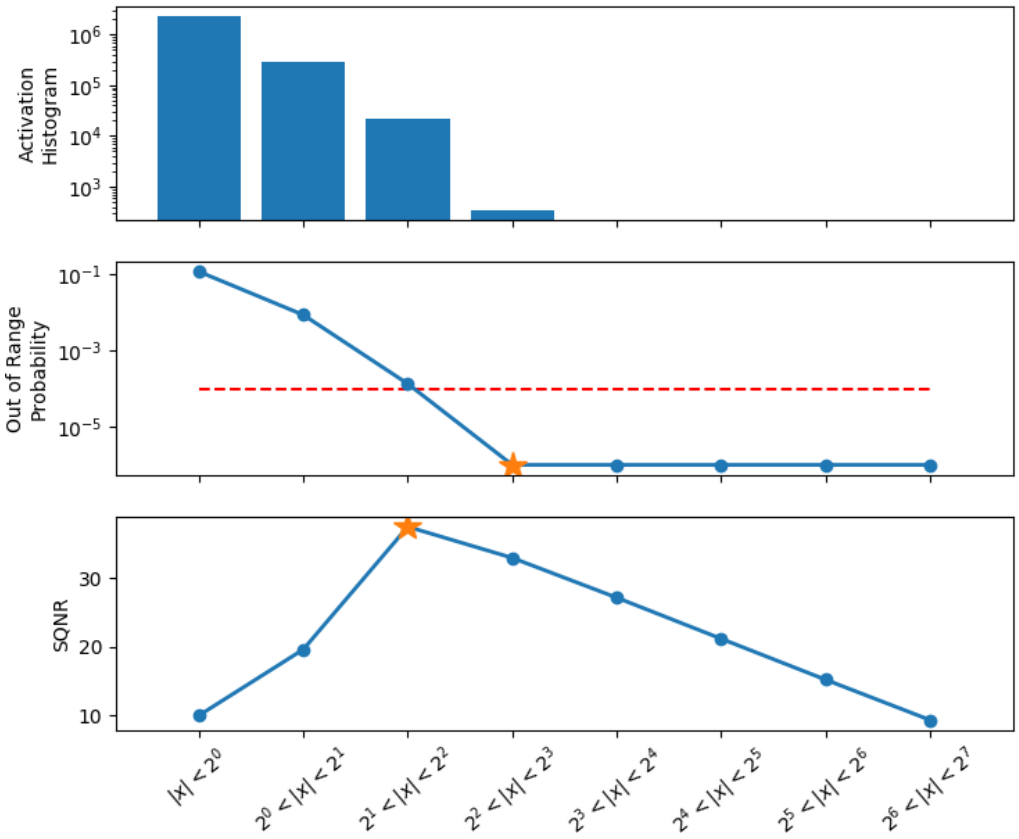

FIGURE 3.6: The upper part shows a histogram of the activation values of one layer, between consecutive powers of 2. The lower part shows an example of the behaviour of the two quantization schemes investigated here. The central graph concludes that 3 decimals should be used for the integer part, because it is the first point in which probability goes below the threshold. The bottom line shows that two digits for the integer part are the solution with maximum SQNR and therefore the minimum MSE

$0 \leq |x| < 2^0$) and 7 bits for the decimal part. In this particular case, corresponding to a layer of the network, the optimum number of bits for the integer part obtained by maximizing the SQNR (three) does not correspond to what obtained by setting the probability threshold to 10^{-4} (four).

3.4.3 Firmware Programming

To transfer the neural model from to CMSIS-NN we developed an automatic python script to export the Keras model in an header file containing the weights to use in the firmware. To do so, firstly, we reordered the weights following the convention of both source and destination framework, then we quantized the weights using both quantization schemes. In this work, we set *bitwidth* to 8 since we privilege execution time and power consumption with respect to accuracy. Note that we implemented a slightly different version of the M_{20k} model, replacing the GRU layer with a vanilla recurrent neural network (RNN). The reason is that the implementation of a GRU layer in CMSIS-NN requires an additional effort while the accuracy does not change significantly.

The CMSIS-NN functions require different arguments: input dimension, number of channel and so on. These parameters should be extracted from the neural network and loaded in the microcontroller. Some of them requires extra processing, like shifting of weights and bias in internal operations. A detailed description of these parameters follows here.

TABLE 3.5: buffers size required for each layer. Reuse of buffers allows memory saving. Two buffers (A,B) must contain maximum value in odd index and even index of $size_{output}$ respectively.

	I	C1	C2	C3	C4	C5
$size_{output}$	6144	**23312**	**10440**	4368	720	...
buffer	A	**B**	**A**	B	A	B

Suppose that we are in an intermediate layer i and that the quantization procedure concludes that the number of integer bits m and the number of decimal bits n for weights, bias, input and output are m_w, n_w, m_w, m_b, m_i, n_i, m_o, m_o. The neural network operations include always a linear combination of weights and input, so that

$$output = input * weight + bias.$$

In multiplications between fixed point numbers, the number of decimal digits of the result are given by the sum of the number of decimal digits of the two operands. To sum the bias to this intermediate value, we need to use the same number of decimals. Thus, we shift the bias to make it match. In most of the cases it is a left shift. Finally, the output must have a certain number of decimals to get back in a fixed bitwidth format, so we apply a further shift. This last shift is opposite to the previous one and in CMSIS-NN is referred to as right shift. These concepts are expressed in formulas, implemented in the header files by means of a macro:

$$left\_shift = (n_i + n_w) - n_b$$
$$right\_shift = (n_i + n_w) - n_o$$

Another parameter to bear in mind is the size of the buffers. To necessarily instantiate buffers in the program, they need to be known in advance. During inference, intermediate values are discarded, and only the final prediction and the network state (in case of recurrent neural network) are kept. As a consequence, it is possible to use the same buffers in multiple layers.

Each layer needs a pair of buffers able to contain output and input. To find the minimum size for these two buffers, we create a vector with the number of element between layers, $size_{output} = a_0, a_1, a_2, ...$, where a_0 is the input size. For each layer, the 2 buffers will switch their role: in the first convolutional layer, the input will be a_0 and the output a_1. For the next layer, a_1 is the input, a_2 is the output and so on. Table 3.5 shows the sequence of input and output for each layer in the implemented network. It shows that the two buffer sizes should be selected accounting for the maximum in odd and even indexes in the vector.

Finally, the CMSIS-NN framework requires also an additional small buffer for intermediate calculations, which can be reused in the implementation too.

3.5 Results

We evaluate the porting of our SED model to the microcontroller, in terms of power consumption, execution and recognition accuracy.

3.5.1 Accuracy

Input data from the test-set are sent by UART to the Microcontroller (MCU) using a Python script. The forward propagation is computed inside the microcontroller that provides the prediction results on the same bus for accuracy evaluation. Following the recipe in [81], we used a 10-fold cross validation and average scores. For each test fold, we load models in the microcontroller with the quantization parameters computed on the related training set.

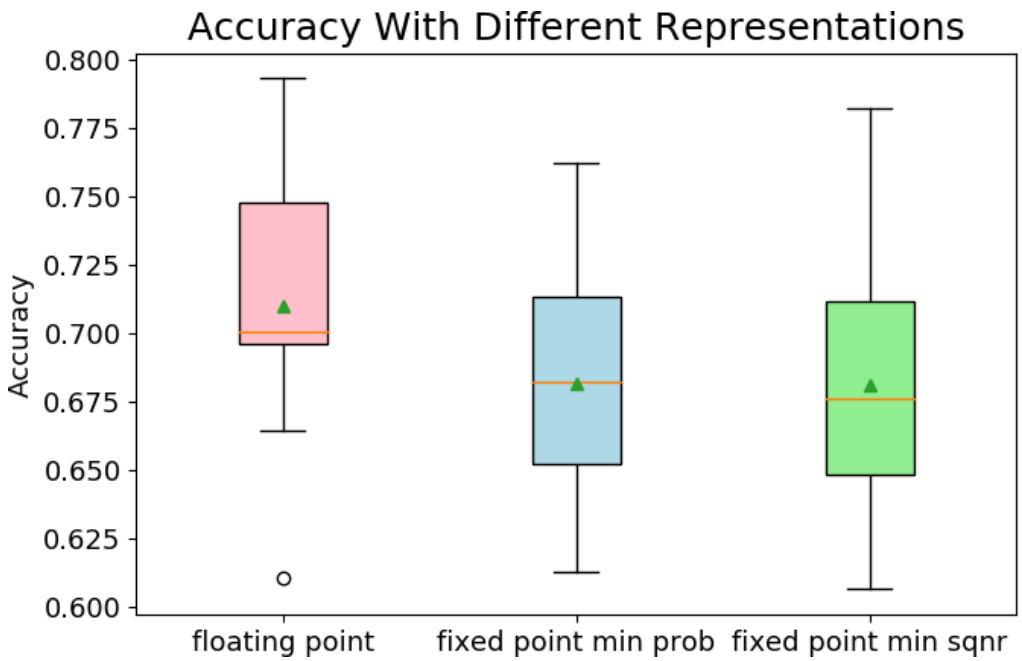

FIGURE 3.7: Accuracy over the 10 folds for *Urbansound8K* dataset. Pink: floating point implementation. Blue: quantization by maximizing the SQNR. Green: quantization by minimizing the probability of overload error

Figure 3.7 depicts the accuracy over the 10 folds. The average accuracy shows a decrease of performance in both quantization schemes, but it is limited to a 2% drop overall if compared to the floating-point version (pink). Note that this minor performance drop could be limited by fine-tuning the quantized network [103].

Looking into more details, this performance deterioration is strongly dependent on the train/test-set split of each fold. Figure 3.8 reports the performance for three folds (train/test configurations): fold 7, fold 10, fold 2. When testing on fold 7, the performance deterioration is in line with the average accuracy decreases of around 2%. On the other hand, when fold 10 is selected as a test set, the accuracy drop is more consistent (8%). Finally, the impact of quantization error becomes negligible, or it even improves the classification results, in fold 2. This behaviour is related to the robustness of the original floating point model. As pointed out by Piczak [70], some classes are more difficult to detect by a convolutional neural network, because of their short scale temporal structure (drilling, engine idling, jackhammer). Whenever the floating point network correctly classifies these classes, the difference between the likelihoods is probably shallow and the errors introduced by quantization can lead to a misclassification, leading to huge performance degradation. Similarly, the gap between the floating point and quantized network decreases or disappear when the reference network already does not classify correctly the difficult classes, as performance is already bad.

To confirm that, we compared the accuracy gap due to quantization and the F1 for these problematic classes. In all three cases, we confirmed that whenever the F1 score is high the accuracy drop increases, following an exponential profile as Figure 3.9 shows. The figure refers to the class "Jackhammer".

3.5.2 Execution Time and Power Consumption

In section 3.2.1, we estimated the execution time of several network architectures by assuming that the number of operations is equivalent to the number of instructions, this way allowing a comparison between different platforms in terms of MIPS available from the datasheets.

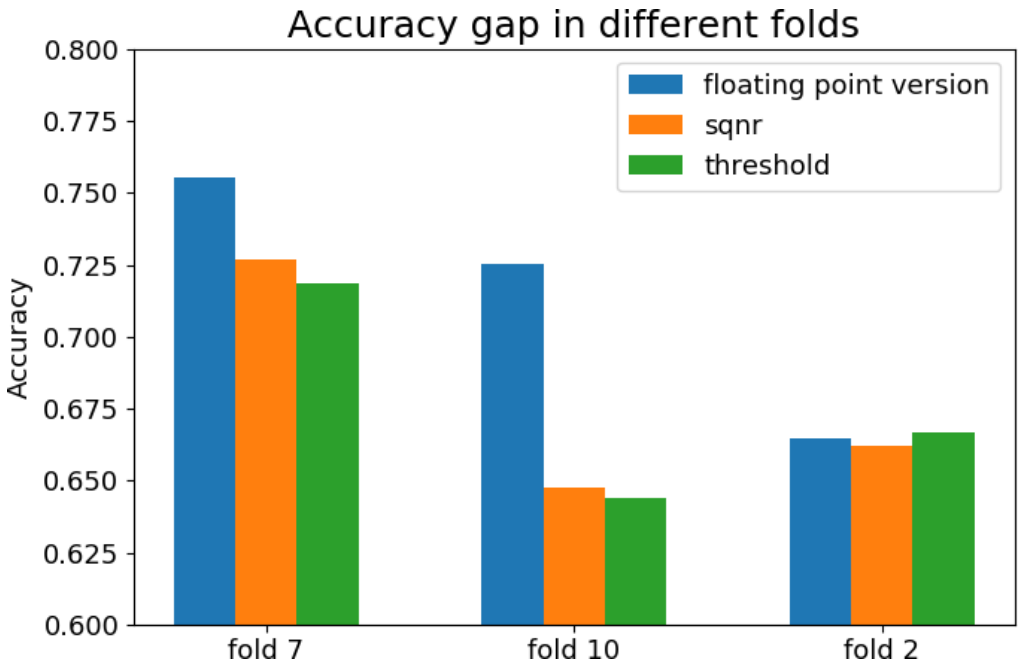

FIGURE 3.8: Accuracy gap between floating point network and quantized versions for different folds. Accuracy degradation in fold 10 is more evident than in fold 2

In this section, we evaluate the actual execution time and the power consumption of our implementation of the smallest model M_{20}. Measurements are done on the real prototype using the same system used to compute the accuracy. We measure the performance layer by layer to understand how each component contributes to the overall processing time. Table 3.6 reports the evaluation results.

The CMSIS-NN framework implements a "basic" and a "fast" version of convolutional layers. The latter uses assembly directives to speed-up execution, especially by means of multiply-accumulate (MAC) and SIMD. The only constraint for using this faster implementation is that the number of channels must be a multiple of 4 for 8-bit fixed-point quantization. This is the reason why, the number of channels of the convolutional layers of all architectures described in Table 3.1 are multiple of 4. Problems arise on the first layer, whose input is the Mel spectrum that has just one channel. This does not allow us to use the optimized version for convolutional layers and for this reason the first convolutional layer takes more than half of overall execution time, with a throughput of 6.56 MOPS (see line 2 of Table 3.6). The second convolutional layer implements the fast version and takes just 20 ms, even if it is more computationally intense, performing 37.34 MOPS. This highlights how important is the parallelization and how much a proper parallelization in the first layer can reduce drastically the overall execution time.

To stress more the importance of an efficient framework for deep neural networks, we used a reference implementation of plain C (without explicit SIMD directives) for convolutional, maxpooling and fully connected layers. The comparison is in the last two lines of Table 3.6. The total execution time is speed-up of x2.32 with the fast implementation of CMSIS-NN with respect to plain C.

In Section 3.3, we stated that the combination of x4 parallelization and overhead, due to branches and load-store instructions, makes the microcontroller able to perform an 8-bit operation in just one instruction. The selected platform executes 80 MIPS, thus we expect 80 MOPS, but the overall throughput is different in real-time measurements (average 17 MOPS). Looking at the peaks in layer conv4, 53.44 MOPS is still far from our estimation

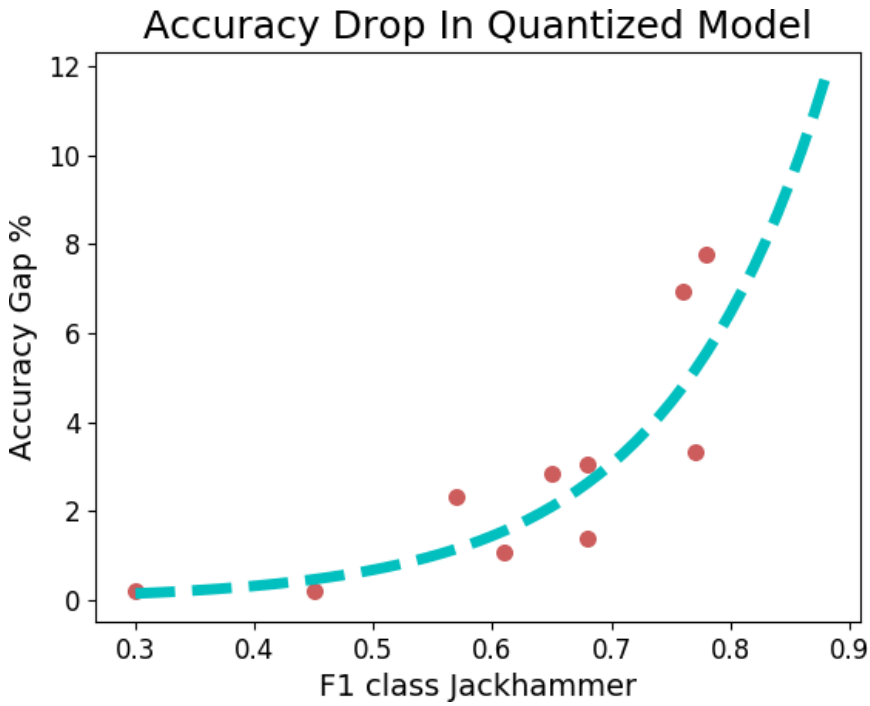

FIGURE 3.9: Relation between performance in a specific class (Jackhammer) and the drop in accuracy in each fold due to quantization error. Due to its short-time duration, the class is difficult to detect for the convolutional model, thus its classification gets harder in the quantized model

of 80 MOPS. It means that parallelization does not fully compensate the overhead due to load-store and branches and we need approximately two instructions to execute an operation. The average (17.08 MOPS) is far from this peak level, mainly because of the first layer, that is the dominant part and it is not fully parallelized.

3.6 Conclusion

To sum up,

- we achieve a student model that, thanks to a two-step knowledge distillation, performs just 2 points below the VGGish baseline in spite of using less than 0.0424% of the parameter and 0.12% of the operation;
- the final implementation on the microcontroller has a propagation time of 125 ms for each 1-second-audio-clip using just 5.5 mW average power and 34.3 kB of RAM.

In this chapter, we continued our exploration of the use of artificial intelligence at the "thing" level, in this case considering audio as the information to elaborate. We described the whole process from a state-of-the-art model for sound event detection to its energy efficient implementation in a microcontroller adequate for IoT applications. In particular, we explored knowledge distillation as a relevant approach to bring accurate context information at the very edge. We also played with the quantization strategy and with the architecture of the student network.

The lesson we learned from the specific case study presented is that knowledge distillation can be effective also for extreme compression rates, achieving models suitable for real time applications on IoT nodes. We also confirmed the lesson learned in the thermal sensor

case study regarding quantization, i.e. that 8-bit quantization is quasi-lossless. In the next chapter, we move the quantization forward to 1 bit per weights and activations. Moreover, the hardware platform, namely GAP8, is a recent commercial product meant to speed up computation of neural networks.

TABLE 3.6: Execution time on the real platform, layer by layer. In the bottom, overall execution time using CMSIS-NN and without DSP optimization (Plain C)

Layer	Output Channel	#kop	Exec. Time [ms]	MOPS
$Input_{96,64}$	1	-	-	-
Conv1	4	419.61	63.93	6.56
Pool1	x	23.31	10.17	2.29
Conv2	8	751.68	20.13	37.34
Pool2	x	24.84	8.32	2.99
Conv3	16	628.99	12.90	48.76
Pool3	x	11.09	3.63	3.03
Conv4	16	207.36	3.88	53.44
Pool4	x	2.16	0.69	3.15
Conv5	32	27.65	0.60	46.23
Pool5	x	0.58	0.17	3.31
FC1	64	8.19	0.22	38,10
FC2	128	16.38	0.42	38.73
RNN	60	22.56	0.55	40,58
FC3	10	1.20	0.04	33.33
Total	-	2145.61	**125.6**	17.08
Plain C	-	2145.61	291.4	7.36

Chapter 4

Binary Neural Network for Sound Event Detection

4.1 Overview

In the previous chapters, we presented the quantization of weights and activations as a powerful tool to reduce execution time and memory requirements. Losses due to lack of numerical precision are negligible in all the cases presented. To summarize, quantization to 8-bit precision is a quasi-lossless speed up for neural networks.

The typical evolution toward this direction is less precise weights and activations, such as 4 bits per number, ternary neural network down to binary neural network. In the last case, weights and activations are described by a single bit, saved in 32bit-registers. It enables the processing of 32 numbers with only a single instruction if the hardware supports specific instructions. Unfortunately, the performance drop here is evident, and it goes from 7% to 15% in various applications. Researchers are trying to reduce the gap of performance between full precision and binary networks, and the methods implemented here are state of the art in loss reduction.

Another step forward made in this chapter towards AI-enabled IoT platforms regards the hardware employed. In the two previous chapters, we implemented our approach on commercial microcontrollers. These platforms have a well-supported software development kit and a variety of examples and resources online. However, they are not updated to the latest progress in literature, and the use of neural networks on microcontrollers is a recent field of interest. On the other hand, hardware research is following the progress in algorithms. The example of a state-of-the-art platform presented here in this chapter uses a multi-core structure powered by low voltages, following the Parallel Ultra Low Power (PULP) paradigm. In addition, the used platform, namely GAP8, has a peculiar Instruction Set Architecture for Binary Neural Network, which includes popcount instructions.

The challenges addressed in this chapter are different from the previous ones: the research platform makes the development difficult, and we designed the Application Program Interfaces for Binary Neural Network, caring about efficiency aspects and the correct use of hardware features such as parallelization.

The combination of hardware parallelism and ad-hoc instructions is an enabling factor for the specific application presented in this chapter. Experimental evaluation shows that our implementation on the PULP platform is 51x more efficient and 10x faster than the implementation of the same network in the Cortex-M4 based counterpart.

4.2 Feature Extraction and BNN

The idea behind BNNs is to approximate the multi-bit filter weights and inputs with binary values in NNs. Binary weights and activations imply a significant decrease in memory usage

as well as computational cost [75]. In this section, we describe the structure of the network, starting from the audio stream to the final prediction.

4.2.1 Feature Extraction (Mel Bins)

The preprocessing part computes the short-time Fourier transform (STFT) in windows of 32 ms every 8 ms. Then, we apply the Mel filters to generate 64 Mel bins. The 400 features are then assembled to create the Mel-spectrogram for 3.2 s of audio. The resulting matrix with a shape of 64×400 is the input to the neural network.

4.2.2 First Layer and Binarization

The input data to the network is non-binary and has, therefore, to be treated separately. A robust approach is to keep the first network layer in full-precision, like in Courbariaux et al. [15]. In this way, the network learns the binarization function from the training set.

After the convolution, batch normalization is applied, which can be replaced in inference by a bias and a scaling factor, and is finally followed by the signum activation function for binarization.

To avoid floating-point operations, all the operations described in this section are done in fixed-point. Fixed-point operations are more efficient in terms of execution time and energy consumption without significant loss of performance [50].

On the other hand, fixed-point quantization requires additional effort in finding the correct amount of integer and fractional bits for each parameter representation. For doing this, we check the range of the parameters, and we choose the number of integer decimals that represents most of the numbers (99.9%) without overload error.

4.2.3 Binary Convolution

BNNs constrain weights and inputs to $\mathbf{I} \in \{-1,1\}^{n_{in} \times h \times b}$ and $\mathbf{W} \in \{-1,1\}^{n_{out} \times n_{in} \times k_y \times k_x}$. To avoid using two bits, we represent -1 with 0, whereas the actual binary numbers are indicated with a hat (i.e., $\hat{i} = (i+1)/2$). It turns out that multiplications become `xnor` operations $\bar{\oplus}$ [75]. Formally the output o_k of an output channel $k \in \{0, ..., n_{out}-1\}$ can be described as[1]:

$$\mathbf{o_k} = \operatorname{sgn}\left(\sum_{n=0}^{n_{in}-1} \mathbf{i_n} * \mathbf{w_{k,n}}\right) = \operatorname{sgn}\left(\sum_{n=0}^{n_{in}-1} 2\left(\hat{\mathbf{i}}_\mathbf{n} * \hat{\mathbf{w}}_\mathbf{k,n}\right) - k_y k_x\right)$$

$$= \operatorname{sgn}\left(\sum_{n=0}^{n_{in}-1} \sum_{(\Delta x, \Delta y)} 2\left(\hat{i}_n^{y+\Delta y, x+\Delta x} \bar{\oplus} \hat{w}_{k,n}^{\Delta y, \Delta x}\right) - 1\right)$$

Whereas Δy and Δx are the relative filter tap positions (e.g., $(\Delta y, \Delta x) \in \{-1,0,1\}^2$ for 3×3 filters). As calculating single-bit operations on microcontroller is not efficient, we pack several input channels into a 32-bit integer (e.g., the feature map pixels at $(y+\Delta y, x+\Delta x)$ in spatial dimension and input channels $32n$ to $(32(n+1)-1)$ packed in $\hat{\mathbf{i}}_{\mathbf{32n}:+\mathbf{32}}^{y+\Delta y, x+\Delta x}$), while the Multiply Accumulates (MACs) can be implemented with *popcount* and *xnor* operations.

Furthermore, as common embedded platforms like GAP8 do not have a built-in *xnor* operator, the *xor* operator $\oplus$ is used and the result is inverted. Therefore, the final equation for the output channel is $\mathbf{o_k} =$

$$\operatorname{sgn}\left(\sum_{n=0}^{\frac{n_{in}}{32}-1} \sum_{(\Delta x, \Delta y)} 32 - 2\operatorname{popcnt}\left(\hat{\mathbf{i}}_{\mathbf{32n}:+\mathbf{32}}^{y+\Delta y, x+\Delta x} \oplus \hat{\mathbf{w}}_{\mathbf{k,32n}:+\mathbf{32}}^{\Delta y, \Delta x}\right)\right)$$

[1] For simplicity, we omit bias and scaling factor in the formula.

TABLE 4.1: Kernel size, channel, and computational effort for each layer.

Layer	Kernel Size	Channel	Stride	MACs
First (real-valued)	3×3	32	1	7M
1. Binary Layer	3×3	64	2	109M
2. Binary Layer	3×3	128	1	405M
3. Binary Layer	3×3	128	2	186M
4. Binary Layer	3×3	128	1	154M
5. Binary Layer	1×1	128	1	17M
Last (real-valued)	1×1	28	1	6M
Total:				884M

4.2.4 Batch Normalization and Binarization

A batch normalization layer follows each binary convolutional layer. As the output of binary layers are integer values, and the signum function can be written as a comparison function, the activation function is simplified to:

$$\cdot \,\mathrm{binAct}(x) = \begin{cases} 0, & \text{if } x \cdot \mathrm{sgn}(\gamma') \geq \left\lfloor \frac{\beta'}{\gamma'} \right\rfloor \\ 1, & \text{if } x \cdot \mathrm{sgn}(\gamma') < \left\lfloor \frac{\beta'}{\gamma'} \right\rfloor \end{cases}. \tag{4.1}$$

whereas γ' is the scaling factor and β' is the bias based on the batch normalization parameters. While exporting the model, we compute the integer threshold value $\lfloor \frac{\beta'}{\gamma'} \rfloor$ in advance. In inference, one sign comparison and one threshold comparison have to be calculated for each activation value.

4.2.5 Last Layer and Prediction

In the last layer, the fixed-point values from the last binary layer are convolved with the fixed-point weights, and N output channels are calculated, where N is the number of classes. Finally, the network performs an average pooling over the whole image giving N predictions for each class.

4.2.6 Neural Network Architecture

Tbl. 4.1 summarizes the architecture of the NN. The neural network consists of 7 hidden layers, 5 of which are binary. The first and last layers are real-valued. Their required computations are significantly smaller than in the binary layers (e.g., 7 MMAC in the first layer compared to 109 MMAC in the second layer), and therefore they minimally contribute to the overall computational effort. The reason for having real-valued layers is the high loss of accuracy with entirely binarized neural networks [75].

4.3 Embedded Implementation

The Mel bins extraction and BNN are implemented on GAP8. The application scenario for this device is low-latency low-power signal processing. The device has a tunable frequency and voltage supply. Fig. 4.1 shows the main block of the chip: GAP8 has two main programmable components, the fabric control (FC), and the cluster. The FC is the central microcontroller unit, and it is meant to manage peripherals and offload workloads to the

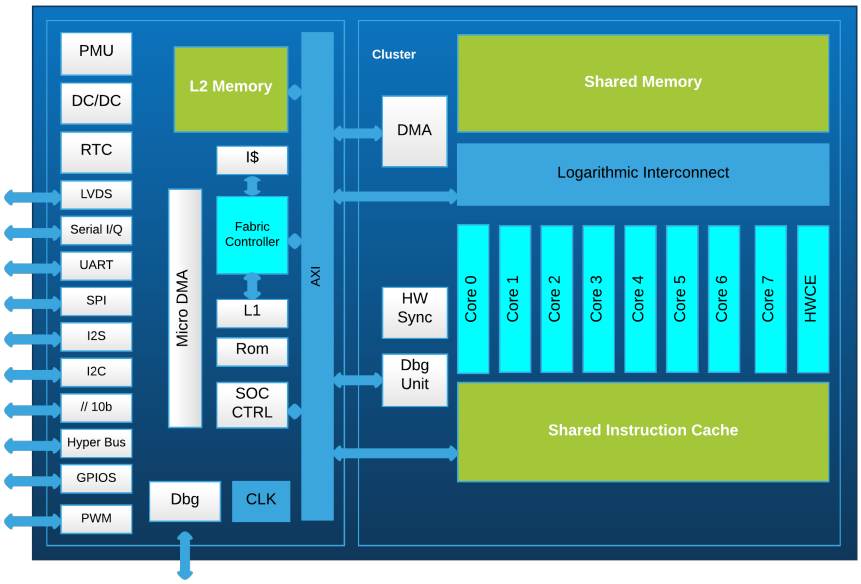

FIGURE 4.1: Architecture of GAP8 embedded processor [21]

cluster. The cluster is composed of eight parallel RISC-V cores, a convolution accelerator, and shared memory banks. The two domains share the same voltage source but keep two different frequencies: On-chip DC-DC converters translate the voltage, and two independent frequency-locked loops (FLLs) generate the two different clock domains. The FC is a single-core in-order microcontroller implementing the RISC-V instruction set. To customize the core for signal processing application, GAP8 extends the RISCV-IMC instruction set for signal processing application. In addition to integer, multiplication, and compressed instruction (IMC), GAP8 ISA supports Multiply and Accumulate, Single Instruction Multiple Data (SIMD), Bit manipulation, post-increment load/store, and Hardware Loops. The FC is directly interconnected to an L2 memory of 512 kB SRAM. The cluster has eight cores identical to the FC. The cores share the 64 kB L1 SRAM scratchpad memory, equipped with a logarithmic interconnect that supports single-cycle concurrent access from different cores requesting memory locations on separate banks.

The cores fetch instructions from a multi-ported instruction cache to maximize the energy efficiency on the data-parallel code. Moreover, an efficient DMA (called µDMA) enables multiple direct transfers from peripherals and L1 to the L2 memory. The cluster has a hardware synchronizer for event management and efficient parallel threads dispatching. The FC and cluster communicate with each other by an AXI-64 bidirectional bus. The software running on the FC overviews all tasks offloaded to the cluster and the µDMA. At the same time, a low-overhead runtime on the cluster cores exploits the hardware synchronizer to implement shared-memory parallelism in the fashion of OpenMP [14].

4.4 Experimental Results

To accurately evaluate the BNN, we designed a full system. Thus, the power and energy-efficient measurements are performed on the hardware platform.

TABLE 4.2: Accuracy and Memory Footprint for the Baseline CNN (16-bit Fixed-Point precision), BNN with first/last layer in 16-bit Fixed-Point.

	CNN [59]	BNN-GAP8
Accuracy	85.1%	**77.9%**
Memory for weights [kB]	815	58
Memory for input [kB]	204	204
Memory requirement [kB]	1019[3]	262

4.4.1 Dataset

For the work presented in this chapter, we use the dataset of Takahashi et al. [90], which is based on the Freesound database, an online collaborative sound database. It consists of 28 different event types, e.g., instruments, animals, mechanical sounds. Each clip has a variable length, and the total length of all 5223 audio files is 768 minutes. All audio samples have a sampling rate of 16 kHz, a bit depth of 16, and are single-channel. The dataset is split into training (75%) and test set (25%). We compute the STFT in windows of 512 samples every 128 samples, respectively 32 ms and 8 ms. Then we apply 64 Mel-filters to generate 64 Mel bins. 400 features are then tiled together to create the Mel-spectrogram for 3.2 s of audio (see Sec. 4.2.1). For the training set, we split each audio clip in consecutive chunks of 3.2 s.

Chunks shorter than 3.2s are discarded, or zero-padded if it is the only chunk. In the test set, we extract one single patch of 3.2 s, starting from half of the clip.

4.4.2 Firmware Details

To cope with L1 memory constraints, we run the prediction on 4 tiles in which the image is split. The tiles have an overlap of 20 pixels to take into account the receptive field of convolutional kernels at the border of the tiles. The firmware implements a double buffering for the weight loading: before the program processes the input of a specific layer, the cores configure the DMA to load the weights of the next layer, from the L2 memory to the single-cycle accessible L1 memory. An interesting feature of GAP8 is the built-in `popcount` instruction, which takes just one cycle and decreases the execution time significantly in binary layers, thus useful for BNN calculation. The single $3\times3\times$C kernel application gains speed thanks to loop unrolling. Finally, the code parallelization over the eight cores is implemented using the OpenMP API.

4.4.3 Accuracy

We start from MeyerNet [59] and use the Additive Noise Annealing (ANA) algorithm [89] to train the network with binary weights and activations. Tbl. 4.2 provides an overview of the original MeyerNet and the BNN. The BNN-GAP8 network keeps the first and the last layer in 16-bit fixed-point, whereas the other layers are binary. For the accuracy of Meyernet, we consider its 16-bit quantized version because it is expected[2] to be the same the FP32 baseline.

The BNN achieves an accuracy of 77.9%, which is 7.3% below the full-precision baseline and is in-line with state-of-the-art binary and ternary networks (i.e., 12% binary and 6.5% ternary neural networks for ImageNet [108, 89]).

Tbl. 4.2 shows that the BNN matches with the memory constraints of 512 kB of L2 memory in GAP8 chip, in contrast to the fixed-point baseline.

[2] DNNs are robust to quantization down to 16 bit [45, 67, 41]

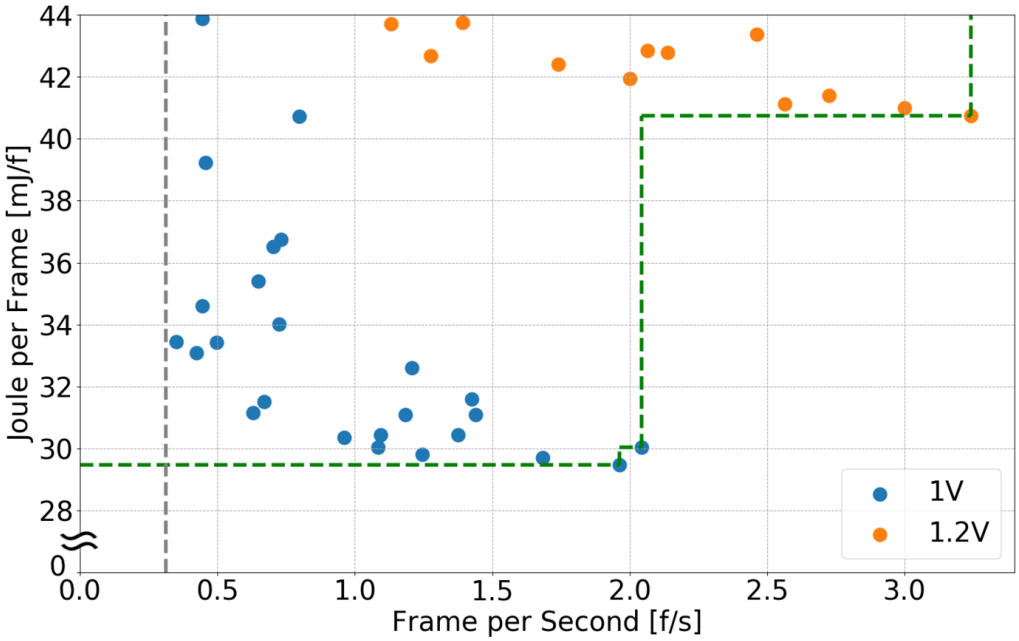

FIGURE 4.2: Throughput and energy efficiency at different supply voltages and operating frequencies. All of the measured settings fulfill the requirement of one classification every 3.2s (see the grey dashed line).

4.4.4 Energy Efficiency

In the following section, we are discussing the throughput and energy efficiency trade-off. First, we sweep the independent cluster and fabric control frequency $(f_{cl}, f_{fc}) \in \{30, 50, 85, 100, 150\}$ MHz $\times$ $\{10, 30, 50, 100, 150\}$ MHz for 1 V, and $(f_{cl}, f_{fc}) \in \{50, 100, 150, 200,$
$250\}$ MHz $\times$ $\{10, 30, 50, 100, 150\}$ MHz for 1.2 V, supported by the GAP8 microcontroller. We set the real-time constraint to 0.3125 frames per second due to the 3.2 s long audio samples.

Fig. 4.2 shows clearly that the 1.0 V corners pareto-dominate the faster 1.2 V corners. It can be seen that the most energy-efficient corner is at 100 MHz for the FC, and 150 MHz for the cluster, where the system achieves an energy efficiency of 31.3 GMAC/s/W, and a throughput of 1.5 GMAC/s.

4.4.5 Execution Time and Power Consumption

We profile time and throughput as well as the energy-efficiency of each layer of the Neural Network (NN). The network architecture is shown in Tbl. 4.1 together with the amount of MAC required for each layer at the most energy-efficient corner according to the analysis in the previous section (i.e., $V_{dd} = 1.0\,\text{V}$, $(f_{cl}, f_{fc}) = (150\,\text{MHz}, 100\,\text{MHz})$).

The measurements are performed with the *Rocketlogger* [86]. Voltage and current of the system-on-chip (SoC) are logged. We evaluate the power and duration of measurements and calculate the energy consumption. The results for each layer are listed in Tbl. 4.3.

Binary layers are the most efficient ones; this is because of the combination of `xor` and `popcount` instructions processing 32 pixels in just 2 instructions. The efficiency peak is at 67.1 GMAC/s/W in the fourth binary layer, and the average efficiency is 34.5 GMAC/s/W. The most efficient configuration meets the real-time constraint, and the entire network runs within 0.511 s.

TABLE 4.3: Duration and energy consumption for each layer as well as throughput and energy efficiency compared to MACs.

Layers	**MACs**	**Time** [ms]	**Energy** [mJ]	**Through.** [MAC/s]	**Efficiency** [MAC/s/W]
Mel bins	-	77.0	2.64	-	-
First Layer	7M	130.8	5.94	54M	1.2G
1. Bin Layer	109M	73.3	3.57	1494M	30.6G
2. Bin Layer	404M	168.0	8.86	2404M	45.6G
3. Bin Layer	185M	51.2	2.94	3628M	63.2G
4. Bin Layer	154M	40.3	2.29	79M	67.1G
5./6. Layer[4]	21M	47.4	1.93	1724M	1.9G
Total/Average	882M	588.0	28.18	1503M	31.3G

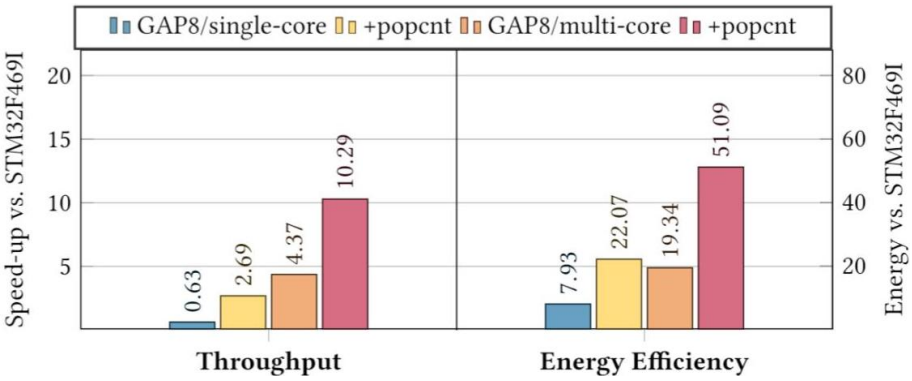

FIGURE 4.3: Improvement in throughput and energy efficiency compared to the ARM Cortex-M4 implementation.

For a further investigation of the improvement in throughput and energy efficiency thanks to the capabilities of the GAP8 SoC, we have implemented the BNN on the STM32F469I Discovery board. Fig. 4.3 gives an overview of the improvements of the GAP8 implementations compared to the single-core ARM Cortex-M4F implementation, which has `popcount` implemented in software. We port the SW-`popcount` (i.e., 12 cycles) to GAP8 and run the code on a single core, and all 8 cores. The GAP8 compared to the STM32F469I, running both the BNN on a single-core and without HW-`popcount`, shows a 7.9× better energy efficiency, but with a 1.6× lower throughput due to the higher operating frequency of the ARM core. Enabling the HW-`popcount` gives a significant improvement in energy efficiency (2.8×) and speed in computation (4.3×). Running the BNN on all 8 cores gives an improvement of 6.9/2.4× in throughput and energy efficiency. Finally, the `popcount` ISA extension gives another boost of 2.4× and 2.6×, respectively.

Overall the GAP8 implementation that uses all the functionality of the core (i.e., `popcount` instruction and multi-core) is 10× faster and 51× more efficient than running the same network on the Cortex-M4F.

Fig. 4.4 shows the power trace of the layers in the same setup in Tbl. 4.3. As described in Sec. 4.4.2, we split the input data into tiles to match the memory constraints. The traces refer to one tile out of four. Thus the execution time is approximately one-fourth of the one presented in Tbl. 4.3. Between layers, the FC offloads the cluster for configuring the next layer: it switches the input and output buffer, allocates memory for the next weights, configures the DMA, and so on. This behavior is visible in the drop of power traces because

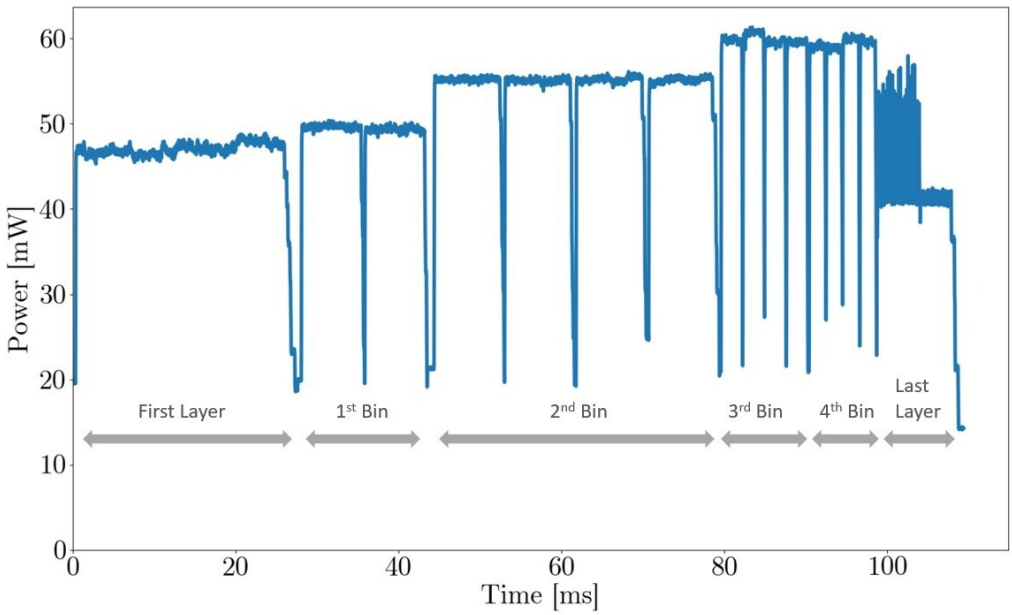

FIGURE 4.4: Power trace of running the BNN on one tile on the GAP8 platform.

the cluster is in sleep, and the activity of the FC consumes less. Similar behavior can be observed inside binary layers, where the processing is split in chunks of 32 channels.

4.5 Conclusions

Starting from the best-performing DNN for sound event detection on our target dataset, we have proposed and trained a DNN with the same topology but binary weights and activations. The proposed BNN matches the memory and resource-constraints of milliwatt range of the target embedded platforms.

The resulting BNN has an accuracy of 77.9%, a drop of 7.2 percent point from the full-precision baseline which is in line of similar state-of-the-art BNNs/TNNs (i.e., 6.5-19%). The overall program requires 230 kB of RAM, 3.9× less than the system using 16-bit quantized baseline CNN. We evaluated energy efficiency with experimental measurement of the power consumption of the full system. The classification of 3.2 s of audio requires 511 ms and 25.54 mJ, with a peak energy efficiency of 67.1 GMAC/s/W and average 34.5 GMAC/s/W. The performance on the GAP8 board has been shown to be 10× faster and 51× more energy-efficient than on an ARM Cortex-M4F platform, which comes from multi-core capabilities (i.e., 4.3/19.3×), the build-in `popcount` instruction (i.e., 2.4/2.6×).

In line with the approach described in the introduction, in this chapter we aim to move AI on low-power IoT node that performs context information extraction. As always, the best performing network does not fit the microcontrollers resources, and here we use BNNs as a compression technique. Furthermore, hardware is the enabling factor to respect the real time constraint.

Moving from the conclusion that GAP is the right hardware for BNNs, the next chapter presents a further step towards low-precision computing. While here the input is a full-precision Mel-frequency spectrogram, in the next chapter the features are generated in hardware. They are again approximated to one-bit resolution, but instead of wasting energy in building the full-precision spectrogram and then quantize it, in the next chapter we directly generate the binary features, saving power in the acquisition and data transfer.

Chapter 5

In-Hardware Audio Feature Extraction for Efficient Binary Neural Network

5.1 Overview

The thermal sensor used in chapter 2 outputs a very low-resolution image; as a consequence, we outlined that the Convolutional Neural Network applied is shallow. In addition, looking at the power consumption, most of the energy is used for the sensor data acquisition. The algorithm itself uses only one-third of the overall power consumption. Similar considerations were driven for the sound event detector in chapter 3. The network's compression is the enabling factor for the embedded implementation because the state of the art network requires too much memory and computational resources. However, after the quasi-lossless compression of the model, microphone reading and feature extraction of the spectrogram use most of the system's energy.In the case of Binary Neural Network (chapter 4), we implemented the feature extraction efficiently, but still, the sensor consumption and most of all the sensor reading takes a considerable amount of energy. All these considerations can be summarized in this way: beside neural network compression, to obtain further power consumption reduction, the focus should be moved to a more efficient data acquisition. The inspiration came from an analog front end for underwater monitoring developed by a group of researchers in ETH. The hardware has suitable characteristics to be reused for audio classification tasks. Here the innovation is the use of binary audio features for binary neural network. More specifically, the analog front end collects data from a microphone and it mixes filtering, amplification, and binarization to output binary signals, which indicates if a certain bandwidth is present in the signal and for how long. Moreover, the analog front end is event-based: it means the hardware wakes up the microcontroller when a generic source of sound is present. This event-based behavior allows a more efficient management of the system activations. Commercial microphones are already low power, but they require amplification and high sampling rate to acquire meaningful data. With the ETH analog front end, there is no sampling paradigm. Whenever a certain frequency bandwidth is present, the microcontroller records the length of the pulse. In this chapter, we demonstrate that this information is already sufficient to perform audio classification. The challenges in this exploration are many because both classification and sensing hardware are based on approximation and it is hard to handle the infinite hyperparameters for such not-established approach. Finally, we present the power consumption and accuracy compared with well-established algorithm for classification. We conclude that the final system can reach less than one milliwatt consumption, using two-order of magnitude less energy than the baseline.

To summarize, the contributions of this chapter are the following:

- we present an innovative use of an analog front end for audio classification, focusing on Keyword Spotting (KWS). We show an in-depth analysis of classification performance and power consumption for different hardware choices. The analog front end reduces energy consumption due to acquisition and processing, and it works as the first-stage able to trigger the full system when sounds of interest are present.
- we present a novel implementation of Binary Neural Network for KWS. The results of a standard solution with full-precision Mel-spectrogram are here compared with a binarization of such features and, finally, with the use of the analog front end. We conclude that Mel-spectrogram and analog front end present similar accuracy results when binarized.
- we show that reducing the classification complexity with BNN is a great advantage in terms of power consumption. However, only in combination with the analog front end, the system reaches 0.96 mW of power consumption, 340 times less than the full precision CNN and 20 times less than the BNN using full precision Mel-spectrogram as an input.

5.2 Analog Front End

Data acquisition and the following feature extraction is performed using hardware component, with a simplified version of the architecture described by Mayer et. Al [54]. Figure 5.1 shows the circuit schematic. The analog signal processing allows the use of any analog microphone, here a low-noise MEMS from InvenSense (ICS-40310) is selected, which consumes only 16 µA at 0.9 V.

The overall aim of the circuit is to extract frequency-time features, so the next step is the filtering of the frequency band of interest. The output of the microphone is connected to a General Impedance Converter (GIC), which uses the OPA379 operational amplifier from Texas Instrument. It features a Gain Bandwidth Product of 90 kHz and a current of 2.9 µA at 1.8 V. The configuration generates a band-pass filter with a center frequency and corner frequency as follow:

$$f_c = \frac{1}{2\pi RC} \quad BW = f_c R_a R_b \tag{5.1}$$

To keep track of the temporal duration for which the frequency is active, the Analog Front End has an active envelope detector connected to the filter's output. It also amplify the output of the GIC, following the formula

$$G = \frac{R_5}{R_4} \tag{5.2}$$

Finally the envelope pass through a comparator, so that the output is only a binary signal The LPV7215 consumes 580 nA at 1.8 V. The first input is the output of the envelope detector, the second is a predefined reference voltage.

During classification, we need more than a single time-frequency detector to perform keyword spotting. Therefore we change the value of resistors and capacitors in the GIC to select a different center frequency and bandwidth of the filter. The same microphone is then connected to different filters, but to avoid overloading the device, we connect a maximum of 8 filters per microphone. If the classification needs more than 8 filters, we add a microphone and the corresponding time-frequency detector. Table 5.1 shows the power consumption using a different number of filters.

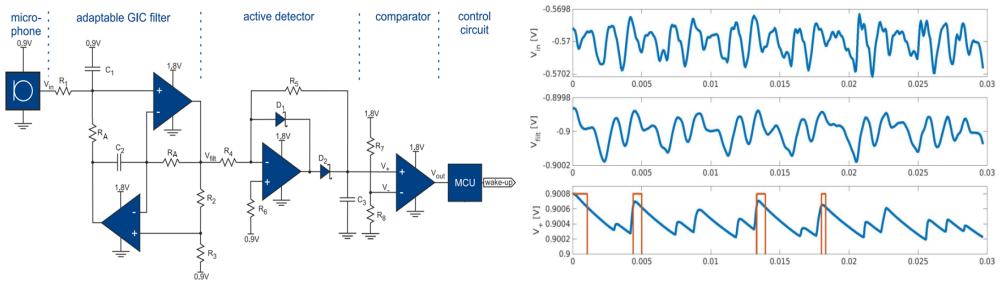

FIGURE 5.1: Left: schematic of the analog front end, divided in its three main blocks. Right: intermediate signal in the circuit. From the top: the microphone output, the filtered signal and the output of the active detector. In red the binary output after the comparator.

TABLE 5.1: Power Consumption. The power consumption of the single elements is depicted in the work of Mayer et al. [54]

System	Power [µW]
Microphone	36
Time-Frequency Detector	15
8 filters AFE	153
16 filters AFE	306
64 filters AFE	612

5.3 Data Acquisition

5.3.1 Firmware

The output of the analog front end contains two pieces of information for each filter. It shows a given frequency component in the input data and shows how long it lasts. The binary output gives the presence; the length of the pulse gives the duration. For the other classification part, we want to save this information. The microcontroller has each digital output connected to a GPIO. In a silent situation, the microcontroller is in sleep mode. When the first interrupt from one of the channels arrives, the microcontroller starts a timer, which will be considered time 0. From now on, the microcontroller saves the timestamp of each interrupt for all the channels. After one second, the firmware starts the reconstruction of the time-frequency representation from the collected data. As in a spectrogram, we divide the time axis in windows of 10 ms, and for each channel and for each window, we fill the time-frequency representation with a "1" if the analog front end output is high at least once inside the window. In this way, the binary image will be the input for the Binary Neural Network.

5.3.2 Software Simulation

To train the neural network, we need to convert audio files to the analog front end outputs and gather them together like the firmware implementation.

We empirically observed that the voltage amplitude, while a person is speaking, is around 5 mV, so we used this value to rescale the sample in the wav files and the equivalent output from the microphone. The following circuitry is simulated in Simulink, using the Openscape library, while the power consumption informations come from the work of Mayer et al [54]. Finally, we do not pass the signal through the comparator, because we want the system to learn not only weights for classification but also the voltage reference of the single comparators (one for each filter). We save the simulation in the time domain, with the sampling rate

TABLE 5.2: Kernel size, channel, and computational effort for each layer

Layer	Kernel Size	Channel	Stride	MACs
1. Binary Layer	3×3	64	2	109M
2. Binary Layer	3×3	128	1	405M
3. Binary Layer	3×3	256	2	186M
4. Binary Layer	3×3	256	1	154M
5. Binary Layer	1×1	256	1	17M
Last (real-valued)	1×1	28	1	6M
Total:				884M

of the input. The windowing of 10 ms described in section 5.3.1 is implemented here in the following manner: we split the sequence in chunks of 10 ms, then we take the maximum value in the chunk and we save its full precision representation. Only during training, we compare the full precision value with a threshold, so that we can learn this threshold.

5.4 Binary Neural Network

From the simulation on the dataset, we obtain the maximum full-precision values of the envelopes in windows of 10 ms. The binarization thresholds (one for each channel) are initialized with the single-channel average value through all the datasets. The full precision envelopes value is normalized using min-max scaling. Finally, during training the thresholds are trainable. This respects the hardware structure of the analog front end, which once defined the best threshold for classification we change the resistors divider to create the desired voltage reference for the comparator. Convolutions, batch normalization, and binarization are the same described in chapter 4, while the structure has a different number of filters and there is not a full precision layer as the input layer. Tbl. 4.1 summarizes the architecture of the NN.

5.5 Experimental Setup

5.5.1 Dataset

We used the Google Speech Command V2, which contains 105k utterances of 35 words [98]. The neural network model is trained to classify the processed audio in one of the 10 keywords: "Yes", "No", "Up", "Down","Left", "Right", "On", "Off", "Stop", "Go". In addition, there is a class "silence" (i.e., no word spoken) and "unknown" word, which is a subset of the utterances in the remaining 20 keywords. Another setup for different applications is the classification of the audio between "yes", "no" and "unknown", from now on referred to as the 3 classes classification. The train-validation-step split has 80:10:10 ratio and it is done following the recommendations of the dataset.

5.5.2 Effect of Model Binarization

First, we trained a well-established convolutional neural network using standard Mel-frequency cepstral coefficients as input. For the whole chapter, it will be considered as a reference for further results. We use windows of 25 ms and hop size of 10 ms to compute the spectrogram. We apply the Mel transformation using 64 Mel-filters in the range between 125 and 7500 Hz, following the methodology described in Hershey et al. [33]. The number of layers and the

number of channels for each layer is described in table Tbl. 4.1, but the layers are here in full precision and not binary layers. Between each convolution, there is Batch Normalization and ReLU activation. We used Adam optimizer and a learning rate of 10^{-4} with a reduction of a factor of ten when the training is not improving for 10 consecutive epochs.

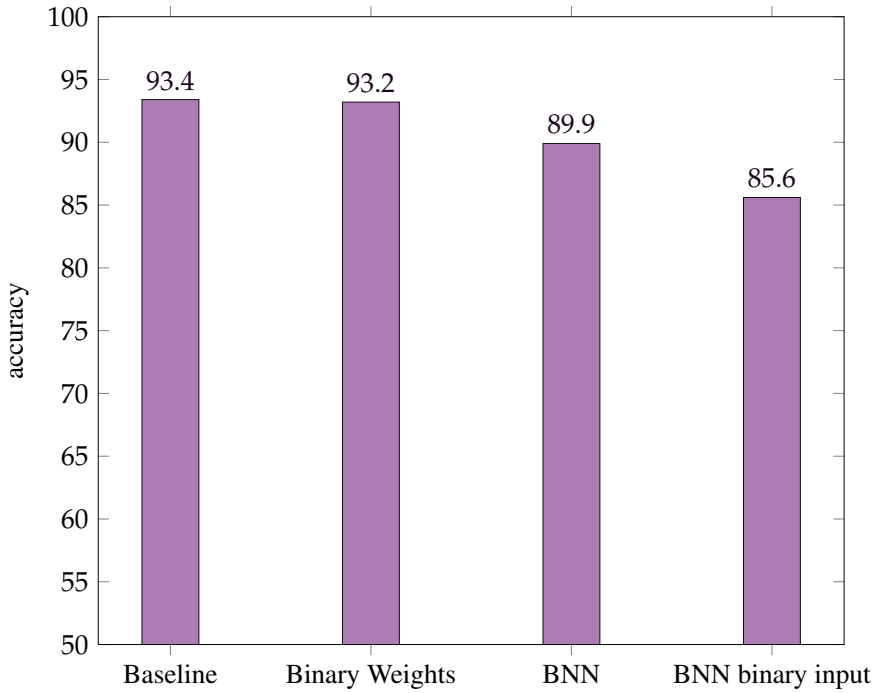

FIGURE 5.2: Effect of binarization of mel-based classifier. The baseline is compared with versions with binary weights, binary weights and activation (BNN), and binary weighs, activation and input (BNN binary input) by using threshold learnt during the training process.

Figure 5.2 shows the degradation performances by using binary numbers instead of full-precision representation. In detail, we binarize three different sets of elements in the network, weights, activations, and input. In the first three cases, the first layer has full precision input, in the last case, with binary input, the Mel-spectrogram is compared with a channel-wise trainable threshold and the first layer receives the binary spectrogram as input. The baseline, where all the data are in full precision, performs slightly worse than the version with quantized weights, most likely because the binarization of weights acts as a regularization in this phase. Quantization of both weights and activations instead drops significantly the performance. Finally, the approach of quantizing the input data using a threshold is the worst case, but it is the most similar scenario to the actual use of the analog front end.

5.5.3 Analog Front End Accuracy Performance

The two paragraphs in Figure 5.3 shows the performance of different configurations in terms of accuracy. They are related to two different tasks: the left one is the classification setup suggested by the creator of the dataset and the right one is the classification of two different keywords ("yes" and "no") and a third classes "unknown", where keywords from the other classes are taken randomly. We use the analog front end with banks of filters: we simulate 8, 16, and 64 filters equally spaced (in the Mel domain) with center frequencies between 150 Hz and 7000 Hz. While the full precision network definitely outperforms all the binarized version, we see that the 64 filter configuration has a very similar performance to the binarized

MFCC classifier. The general trend is that by reducing the number of filters the performance gets worse. This matches the expectations, as the amount of information decreases with the with fewer filters.

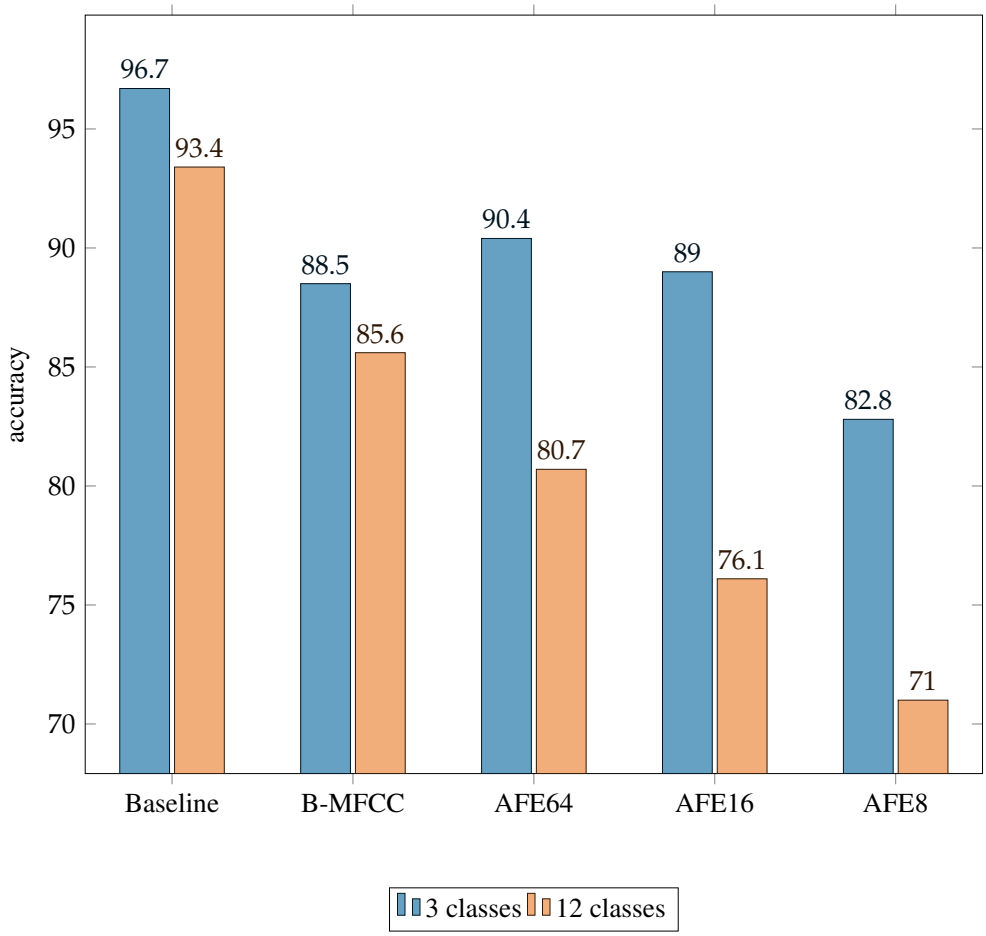

FIGURE 5.3: The baseline is compared with BNN with binary inputs, including Binary Mel bins and the analog front end with 64, 16 and 8 filters. Blue bars refer to "yes", "no" and unknown keywords, Orange bars refer to the standard 10 keyword task plus the classes "silence" and unknown keywords.

5.5.4 Power Consumption

The 5 different configurations have different power consumption. There are three main blocks that can be isolated for power consumption, namely the sensor consumption, the feature extraction and reading of data, and finally classification. For what concerns data reading of the analog front end, Section 5.3.1 describes the firmware used for the acquisition of 1-second data. The consumption of such phase is mainly given by the short Interrupt Service in which the microcontroller saves the timestamp of such events. To estimate this power, we run the firmware with a set of interrupt rates on real hardware and we estimate the power consumption per interrupt. Finally, we compute the number of interrupts in the simulations for each different setup (8, 16 and 64 filters). The platform used for experimentation is the STM32l476RG. Finally, the consumption of the neural network is computed through the values of the energy efficiency of BNN and fixed point layers presented in the chapter 4. Table 5.5.4 shows the consumption per each of the configuration. The power consumption of the

TABLE 5.3: Energy consumption for the different system proposed

	Mel bins	Bin Mel	AFE 64	AFE16	AFE8
Microphone	1.08	1.08	0.61	0.31	0.15
Sensor Reading	5.40	5.40	0.29	0.16	0.13
Classification	320.63	14.27	14.27	3.56	0.68
Total [mJ]	327.11	20.75	15.17	4.02	0.96

full-precision version is around 80 mJ, and the binarization of weights and activation reduces the power consumption of the neural network inference, but not of the acquisition and feature extraction. The analog front end combines the low power analog microphone together, in hardware feature extraction and event-based reading to significantly reduce all the power consumption.

5.6 Conclusion

In this chapter we keep reducing power consumption to move AI-based system on low power IoT nodes. We presented the use of an analog front end to minimize the contribution to power consumption given by specific functionalities of a classification system, namely sensing and feature extraction. we presented how features extracted from the analog front end are as good as binarized Mel-spectrogram for classification using BNN, and they require significantly less power to be acquired. However, the accuracy results are not comparable with a full-precision Convolutional Neural Network that uses Mel-spectrogram. This wide gap in the accuracy (from 93% to 71%) makes this approach not lossless and still requires further development to improve the BNN accuracy performance, as we also concluded in the previous chapter. On top of this exploratory work, we suggest a different trend towards better accuracy than lower power consumption. The feature that shows to decrease the accuracy significantly is a binarization of the input (see figure 5.2); thus, a more advanced approach should extract Mel-spectrogram-like features but keeping the signal in full-precision.

Chapter 6

Conclusions

In conclusion, this thesis presented hardware and software strategies to reduce power consumption and memory footprint for tasks that usually require high-performance hardware and billion of computation, with the main goal of near sensor processing in an IoT infrastructure. Microcontroller implementations of AI-powered systems were rare at the time of the thesis and this field is continuously growing, so this thesis adds one of the first bricks to the foundation of ubiquitous and intelligent devices.

From the research done in the framework of this thesis, some general conclusions can be extracted about present trends in tinyML, or machine learning in IoT end nodes. First of all, Artificial Intelligence in IoT nodes is feasible, without renouncing to high accuracy performance. However, it is not straight-forward but requires different training strategies with respect to networks that target only state-of-the-art accuracy.

Not all the compression techniques are the same and the same technique may behave differently in different fields. The student-teacher approach has demonstrated to be a valid approach to significantly reduce consumption keeping the accuracy loss limited or even negligible. Moreover, this approach can lead to significant complexity reduction and the designer can decide when to stop compressing as soon as the performance starts decreasing. Oppositely, quantization of weight and activation does not always preserve accuracy, and the boost given by quantization is not as significant as the student-teacher approach. The starting structure can be reduced only by a factor of 32, which is not enough in most cases. While quantization is "safe" down to 8 bit per weight and activation, further levels of approximation lead to poor accuracy performance, as shown in the last two chapters.

To summarize, I conclude that quantization is a powerful tool to make computation more efficient but it should be limited to 8-bit resolution and it should be used in combination with other structural changes of the neural network (e.g. Student-Teacher).

Bibliography

[1] *5 Reasons the new Arm Cortex-M55 will transform the IoT*. \url{https://www.beningo.com/5-reasons-the-new-arm-cortex-m55-will-transform-the-iot/}.

[2] Erwin Adi et al. "Machine learning and data analytics for the IoT". In: *Neural Computing and Applications* (2020), pp. 1–29.

[3] Andrawes Al Bahou et al. "XNORBIN: A 95 TOp/s/W hardware accelerator for binary convolutional neural networks". In: *2018 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*. IEEE. 2018, pp. 1–3.

[4] Massimo Alioto. "IoT: bird's eye view, megatrends and perspectives". In: *Enabling the Internet of Things*. Springer, 2017.

[5] Renzo Andri et al. "YodaNN: An Ultra-Low Power Convolutional Neural Network Accelerator Based on Binary Weights". In: *Proc. IEEE ISVLSI*. 2016, pp. 236–241. ISBN: 978-1-4673-9039-2. DOI: 10.1109/ISVLSI.2016.111.

[6] Kevin Ashton and others. "That 'internet of things' thing". In: *RFID journal* 22.7 (2009), pp. 97–114.

[7] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey". In: *Computer networks* 54.15 (2010), pp. 2787–2805.

[8] Mohammad Babaeizadeh, Paris Smaragdis, and Roy H Campbell. "Noiseout: A simple way to prune neural networks". In: *arXiv preprint arXiv:1611.06211* (2016).

[9] Chandrayee Basu and Anthony Rowe. "Tracking Motion and Proxemics Using Thermal-Sensor Array". In: *arXiv preprint arXiv:1511.08166* (2015).

[10] Flavio Bonomi et al. "Fog computing and its role in the internet of things". In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. 2012, pp. 13–16.

[11] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. "An Analysis of Deep Neural Network Models for Practical Applications". In: *arXiv preprint arXiv:1605.07678* (2016). URL: http://arxiv.org/abs/1605.07678.

[12] Phoebus Chen et al. "Instrumenting wireless sensor networks for real-time surveillance". In: *Robotics and Automation, 2006. ICRA 2006. IEEE Int. Conf. on*. IEEE. 2006, pp. 3128–3133.

[13] *City of Things: Let's design a smarter city together*.

[14] Francesco Conti et al. "Enabling the heterogeneous accelerator model on ultra-low power microcontroller platforms". In: *Proceedings of the 2016 Design, Automation and Test in Europe Conference and Exhibition, DATE 2016*. EDA Consortium. 2016, pp. 1201–1206. ISBN: 9783981537062. DOI: 10.3850/9783981537079{_}0626.

[15] Matthieu Courbariaux and others. "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1". In: *arXiv:1602.02830*. 2016.

[16] Jason Cramer et al. “Look, Listen, and Learn More: Design Choices for Deep Audio Embeddings”. In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. Vol. 2019-May. 2019, pp. 3852–3856. ISBN: 9781479981311. DOI: 10.1109/ICASSP.2019.8682475.

[17] James W Davis and Mark A Keck. “A two-stage template approach to person detection in thermal imagery”. In: *Application of Computer Vision, 2005. WACV/MOTIONS'05 7th IEEE Workshops on*. Vol. 1. IEEE. 2005, pp. 364–369.

[18] Yuan Jie Fan et al. “IoT-based smart rehabilitation system”. In: *IEEE transactions on industrial informatics* 10.2 (2014), pp. 1568–1577.

[19] Elisabetta Farella and others. “Technologies for a thing-centric internet of things”. In: *Proc. IEEE FiCloud*. 2017.

[20] P Feng et al. “Deep learning for posture analysis in fall detection”. In: *2014 19th International Conference on Digital Signal Processing*. 2014, pp. 12–17.

[21] Eric Flamand et al. “GAP-8: A RISC-V SoC for AI at the Edge of the IoT”. In: *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE. 2018, pp. 1–4.

[22] Jort F. Gemmeke et al. “Audio Set: An ontology and human-labeled dataset for audio events”. In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. IEEE. Mar. 2017, pp. 776–780. ISBN: 9781509041176. DOI: 10.1109/ICASSP.2017.7952261.

[23] Jort F. Gemmeke et al. “Audio Set: An ontology and human-labeled dataset for audio events”. In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. IEEE. 2017, pp. 776–780. ISBN: 9781509041176. DOI: 10.1109/ICASSP.2017.7952261.

[24] Luigi Gerosa et al. “Scream and gunshot detection in noisy environments”. In: *2007 15th European Signal Processing Conference*. IEEE. 2007, pp. 1216–1220.

[25] Andres Gomez, Francesco Conti, and Luca Benini. “Thermal image-based CNN's for ultra-low power people recognition”. In: *Proceedings of the 15th ACM International Conference on Computing Frontiers*. 2018, pp. 326–331.

[26] *Grid-Eye state of the art thermal imaging solution*. Tech. rep. 2016.

[27] Jayavardhana Gubbi et al. “Internet of Things (IoT): A vision, architectural elements, and future directions”. In: *Future generation computer systems* 29.7 (2013), pp. 1645–1660.

[28] Ivan Gudymenko, Katrin Borcea-Pfitzmann, and Katja Tietze. “Privacy implications of the internet of things”. In: *International Joint Conference on Ambient Intelligence*. Springer. 2011, pp. 280–286.

[29] Awni Y Hannun and others. “Deep Speech: Scaling up end-to-end speech recognition”. In: *Computing Research Repository* abs/1412.5 (2014).

[30] Abbas M Hassan and Ali Ismail Awad. “Urban transition in the era of the internet of things: Social implications and privacy challenges”. In: *IEEE Access* 6 (2018), pp. 36428–36440.

[31] Babak Hassibi, David G. Stork, and Gregory J. Wolff. “Optimal brain surgeon and general network pruning”. In: *1993 IEEE International Conference on Neural Networks*. IEEE. 1993, pp. 293–299. ISBN: 0780312007. DOI: 10.1109/icnn.1993.298572.

[32] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[33] Shawn Hershey et al. "CNN architectures for large-scale audio classification". In: *ICASSP, IEEE*. IEEE. 2017, pp. 131–135. ISBN: 9781509041176. DOI: 10.1109/ICASSP.2017.7952132.

[34] G Hinton et al. "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97.

[35] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the Knowledge in a Neural Network". In: *arXiv preprint arXiv:1503.02531* (2015). URL: http://arxiv.org/abs/1503.02531.

[36] Sang Gi Hong, Nae Soo Kim, and Whan Woo Kim. "Reduction of false alarm signals for PIR sensor in realistic outdoor surveillance". In: *ETRI Journal* 35.1 (2013), pp. 80–88.

[37] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[38] Itay Hubara and others. "Binarized neural networks". In: *Adv. NIPS*. 2016, pp. 4107–4115.

[39] Forrest N Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size". In: *arXiv preprint arXiv:1602.07360* (2016).

[40] M H Ionica and D Gregg. "The Movidius Myriad Architecture's Potential for Scientific Computing". In: *IEEE Micro* 35.1 (2015), pp. 6–14.

[41] Benoit Jacob and others. "Quantization and training of neural networks for efficient integer-arithmetic-only inference". In: *Proc. IEEE CVPR*. 2018.

[42] Lihong Jiang et al. "An IoT-oriented data storage framework in cloud computing platform". In: *IEEE Transactions on Industrial Informatics* 10.2 (2014), pp. 1443–1451.

[43] Jurgen Kemper and Holger Linde. "Challenges of passive infrared indoor localization". In: *Positioning, Navigation and Communication, 2008. WPNC 2008. 5th Workshop on*. IEEE. 2008, pp. 63–70.

[44] Olga Krestinskaya and Alex Pappachen James. "Binary weighted memristive analog deep neural network for near-sensor edge processing". In: *2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO)*. IEEE. 2018, pp. 1–4.

[45] Liangzhen Lai, Naveen Suda, and Vikas Chandra. "CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs". In: *arXiv preprint arXiv:1801.06601* (2018). URL: http://arxiv.org/abs/1801.06601.

[46] M Lecca et al. "A low power smart camera for video surveillance and forensic applications". In: *2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. 2017, pp. 626–631.

[47] In Lee and Kyoochun Lee. "The Internet of Things (IoT): Applications, investments, and challenges for enterprises". In: *Business Horizons* 58.4 (2015), pp. 431–440.

[48] Cong Leng et al. "Extremely low bit neural network: Squeeze the last bit out with ADMM". In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*. 2018, pp. 3466–3473. ISBN: 9781577358008.

[49] Shancang Li, Li Da Xu, and Shanshan Zhao. "The internet of things: a survey". In: *Information Systems Frontiers* 17.2 (2015), pp. 243–259.

[50] Darryl D. Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. "Fixed point quantization of deep convolutional networks". In: *33rd International Conference on Machine Learning, ICML 2016*. Vol. 6. 2016, pp. 4166–4175. ISBN: 9781510829008.

[51] Pedro Garcia Lopez et al. "Edge-centric computing: Vision and challenges". In: *Computer Communication Review* 45.5 (Sept. 2015), pp. 37–42. ISSN: 19435819. DOI: 10.1145/2831347.2831354.

[52] Thomas Lorenser. *The DSP capabilities of ARM Cortex-M4 and Cortex-M7 Processors*. Tech. rep. November. ARM, Nov. 2016, p. 19.

[53] James Manyika et al. *Disruptive technologies: Advances that will transform life, business, and the global economy*. Vol. 180. McKinsey Global Institute San Francisco, CA, 2013.

[54] Philipp Mayer, Michele Magno, and Luca Benini. "Self-sustaining acoustic sensor with programmable pattern recognition for underwater monitoring". In: *IEEE Transactions on Instrumentation and Measurement* 68.7 (2019), pp. 2346–2355.

[55] Qi Meng and others. "The influence of crowd density on the sound environ. of commercial pedestrian streets". In: *Sci. Total Environ.* 511 (2015). ISSN: 18791026.

[56] Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen. "TUT database for acoustic scene classification and sound event detection". In: *European Signal Processing Conference*. Vol. 2016-Novem. Budapest, Hungary, 2016, pp. 1128–1132. ISBN: 9780992862657. DOI: 10.1109/EUSIPCO.2016.7760424.

[57] Annamaria Mesaros and others. "DCASE 2017 Challenge setup : Tasks , datasets and baseline system". In: *Proc. DCASE*. 2017.

[58] Annamaria Mesaros et al. "Acoustic event detection in real life recordings". In: *European Signal Processing Conference*. 2010, pp. 1267–1271.

[59] Matthias Meyer, Lukas Cavigelli, and Lothar Thiele. "Efficient Convolutional Neural Network For Audio Event Detection". In: *arXiv preprint arXiv:1709.09888* (2017). URL: http://arxiv.org/abs/1709.09888.

[60] Daniele Miorandi et al. "Internet of things: {\{}Vision{\}}, applications and research challenges". In: *Ad {\{}Hoc{\}} {\{}Networks{\}}* 10.7 (2012), pp. 1497–1516. ISSN: 1570-8705. DOI: 10.1016/j.adhoc.2012.02.016.

[61] Mehdi Mohammadi et al. "Deep learning for IoT big data and streaming analytics: A survey". In: *IEEE Communications Surveys \& Tutorials* 20.4 (2018), pp. 2923–2960.

[62] David Moloney et al. "Myriad 2: Eye of the computational vision storm". In: *2014 IEEE Hot Chips 26 Symposium (HCS)*. IEEE. 2014, pp. 1–18.

[63] Bert Moons and others. "BinarEye: An always-on energy-accuracy-scalable binary CNN processor with all memory on chip". In: *Proc. IEEE CICC*. 2018.

[64] Yueyue Na et al. "An acoustic traffic monitoring system: Design and implementation". In: *UIC-ATC-ScalCom*. IEEE. 2015, pp. 119–126.

[65] Veli-Pekka Niitamo et al. "State-of-the-art and good practice in the field of living labs". In: *2006 IEEE international technology management conference (ICE)*. IEEE. 2006, pp. 1–8.

[66] Francesco Paissan et al. "People/Car Classification using an Ultra-Low-Power Smart Vision Sensor". In: *2019 IEEE 8th International Workshop on Advances in Sensors and Interfaces (IWASI)*. IEEE. 2019, pp. 91–96.

[67] Daniele Palossi et al. "A 64-mW DNN-Based Visual Navigation Engine for Autonomous Nano-Drones". In: *IEEE Internet of Things Journal*. Vol. 6. 5. 2019, pp. 8357–8371. DOI: 10.1109/JIOT.2019.2917066. URL: http://arxiv.org/abs/1805.01831.

[68] Raymond Perrault et al. "The AI Index 2019 Annual Report". In: *AI Index Steering Committee, Human-Centered AI Institute, Stanford University, Stanford, CA* (2019).

[69] Massimo Piccardi. "Background subtraction techniques: a review". In: *Systems, man and cybernetics, 2004 IEEE Int. Conf. on*. Vol. 4. IEEE. 2004, pp. 3099–3104.

[70] Karol J. Piczak. "Environmental sound classification with convolutional neural networks". In: *IEEE International Workshop on Machine Learning for Signal Processing, MLSP*. Vol. 2015-Novem. IEEE. 2015, pp. 1–6. ISBN: 9781467374545. DOI: 10.1109/MLSP.2015.7324337.

[71] Karol J. Piczak. "ESC: Dataset for environmental sound classification". In: *MM 2015 - Proceedings of the 2015 ACM Multimedia Conference*. ACM. 2015, pp. 1015–1018. ISBN: 9781450334594. DOI: 10.1145/2733373.2806390.

[72] Gopika Premsankar, Mario Di Francesco, and Tarik Taleb. "Edge computing for the Internet of Things: A case study". In: *IEEE Internet of Things Journal* 5.2 (2018), pp. 1275–1284.

[73] Haotong Qin et al. "Binary neural networks: A survey". In: *Pattern Recognition* (2020), p. 107281.

[74] Chakravartula Raghavachari et al. "A comparative study of vision based human detection techniques in people counting applications". In: *Procedia Computer Science* 58 (2015), pp. 461–469.

[75] Mohammad Rastegari et al. "Xnor-net: Imagenet classification using binary convolutional neural networks". In: *European Conference on Computer Vision*. Springer. 2016, pp. 525–542.

[76] Adriana Romero et al. "FitNets: Hints for thin deep nets". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015).

[77] Manuele Rusci. "Ultra-Low Power IoT Smart Visual Sensing Devices for Always-ON Applications". PhD thesis. alma, 2018.

[78] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *IJCV* 115.3 (2015), pp. 211–252. ISSN: 0920-5691. DOI: 10.1007/s11263-015-0816-y.

[79] Tara N. Sainath and Carolina Parada. "Convolutional neural networks for small-footprint keyword spotting". In: *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*. Vol. 2015-Janua. 2015, pp. 1478–1482.

[80] Justin Salamon and Juan Pablo Bello. "Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification". In: *IEEE Signal Processing Letters* 24.3 (2017), pp. 279–283. ISSN: 10709908. DOI: 10.1109/LSP.2017.2657381.

[81] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. "A dataset and taxonomy for urban sound research". In: *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia*. ACM. 2014, pp. 1041–1044. ISBN: 9781450330633. DOI: 10.1145/2647868.2655045.

[82] Shree Krishna Sharma and Xianbin Wang. "Live data analytics with collaborative edge and cloud processing in wireless IoT networks". In: *IEEE Access* 5 (2017), pp. 4621–4635.

[83] Jonathan Shen et al. "In Teacher We Trust: Learning Compressed Models for Pedestrian Detection". In: *arXiv preprint arXiv:1612.00478* (2016). URL: http://arxiv.org/abs/1612.00478.

[84] Yun Q. Shi and Huifang Sun. *Image and video compression for multimedia engineering: Fundamentals, algorithms, and standards, second edition*. CRC press, 2017, pp. 1–541. ISBN: 9781420007268. DOI: 10.1201/9781420007268.

[85] Chi-Sheng Shih et al. "Collaborative Sensing for Privacy Preserving Gait Tracking Using IoT Middleware". In: *Int. Conf. on Research in Adaptive and Convergent Systems*. ACM. 2017, pp. 152–159.

[86] Lukas Sigrist and others. "Measurement and Validation of Energy Harvesting IoT Devices". In: *Proc. IEEE DATE*. 2017.

[87] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *ArXiv:1409.1556* (2014). URL: http://arxiv.org/abs/1409.1556.

[88] Andrews Sobral and Antoine Vacavant. "A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos". In: *Computer Vision and Image Understanding* 122 (2014), pp. 4–21.

[89] Matteo Spallanzani and others. "Additive noise annealing and approximation properties of quantized neural networks". In: *arXiv:1905.10452* (2019).

[90] Naoya Takahashi et al. "Deep convolutional neural networks and data augmentation for acoustic event recognition". In: *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*. Vol. 08-12-Sept. Sept. 2016, pp. 2982–2986. DOI: 10.21437/Interspeech.2016-805.

[91] Mingxing Tan and Quoc V Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *arXiv preprint arXiv:1905.11946* (2019).

[92] Raphael Tang and Jimmy Lin. "Deep residual learning for small-footprint keyword spotting". In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. Vol. 2018-April. 2018, pp. 5484–5488. ISBN: 9781538646588. DOI: 10.1109/ICASSP.2018.8462688.

[93] Andrey Temko et al. "CLEAR evaluation of acoustic event detection and classification systems". In: *Lecture Notes in Computer Science*. Vol. 4122 LNCS. Springer Berlin Heidelberg, 2007, pp. 311–322. ISBN: 9783540695677. DOI: 10.1007/978-3-540-69568-4{_}29.

[94] Sebastian Uziel et al. "Networked embedded acoustic processing system for smart building applications". In: *DASIP*. IEEE. 2013, pp. 349–350. ISBN: 9791092279016.

[95] Rob Van Kranenburg and Alex Bassi. "IoT challenges". In: *Communications in Mobile Computing* 1.1 (2012), p. 9.

[96] VentureBeat.com. *GreenWaves Technologies unveils Gap8 processor for AI at the edge*. 2018.

[97] Van-Thinh Vu et al. "Audio-video event recognition system for public transport security". In: (2006).

[98] Pete Warden. "Speech commands: A dataset for limited-vocabulary speech recognition". In: *arXiv preprint arXiv:1804.03209* (2018).

[99] R W Whatmore. “Pyroelectric devices and materials”. In: *Reports on progress in physics* 49.12 (1986), p. 1335.

[100] Joseph Yiu. *The Definitive Guide to ARM®Cortex®-M3 and Cortex®-M4 Processors*. Newnes, 2013.

[101] Joseph Yiu. *White paper:Introduction to the Arm Cortex-M55 Processor*. Tech. rep. ARM, Feb. 2020.

[102] Manas Kumar Yogi, K Chandrasekhar, and G Vijay Kumar. “Mist computing: Principles, trends and future direction”. In: *arXiv preprint arXiv:1709.06927* (2017).

[103] Linghua Zeng, Zhangcheng Wang, and Xinmei Tian. “KCNN: Kernel-wise quantization to remarkably decrease multiplications in convolutional neural network”. In: *IJCAI International Joint Conference on Artificial Intelligence*. Vol. 2019-Augus. 2019, pp. 4234–4242. ISBN: 9780999241141. DOI: 10.24963/ijcai.2019/588.

[104] Xiaohu Zhang, Yuexian Zou, and Wei Shi. “Dilated convolution neural network with LeakyReLU for environmental sound classification”. In: *International Conference on Digital Signal Processing, DSP*. Vol. 2017-Augus. IEEE. 2017, pp. 1–5. ISBN: 9781538618950. DOI: 10.1109/ICDSP.2017.8096153.

[105] Yundong Zhang et al. “Hello Edge: Keyword Spotting on Microcontrollers”. In: *arXiv preprint arXiv:1711.07128* (2017).

[106] Zhichao Zhang et al. “Deep convolutional neural network with mixup for environmental sound classification”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11257 LNCS. Springer. 2018, pp. 356–367. ISBN: 9783030033347. DOI: 10.1007/978-3-030-03335-4{_}31.

[107] Aojun Zhou et al. “Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights”. In: *Proc. ICLR*. 2017.

[108] Shuchang Zhou and others. “Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients”. In: *arXiv:1606.06160* (2016).

[109] Xiaodan Zhuang et al. “Real-world acoustic event detection”. In: *Pattern Recognition Letters* 31.12 (2010), pp. 1543–1551. ISSN: 01678655. DOI: 10.1016/j.patrec.2010.02.005.