Monocular Depth Perception on Microcontrollers for Edge Applications

Monocular Depth Perception on Microcontrollers for Edge Applications

Valentino Peluso, *Member, IEEE*, Antonio Cipolletta, *Student Member, IEEE*, Andrea Calimera, *Member, IEEE*, Matteo Poggi, *Member, IEEE*, Fabio Tosi, *Student Member, IEEE*, Filippo Aleotti, *Student Member, IEEE*, Stefano Mattoccia *Member, IEEE*

***Abstract*—Depth estimation is crucial in several computer vision applications, and a recent trend in this field aims at inferring such a cue from a single camera. Unfortunately, despite the compelling results achieved, state-of-the-art monocular depth estimation methods are computationally demanding, thus precluding their practical deployment in several application contexts characterized by low-power constraints. Therefore, in this paper, we propose a lightweight Convolutional Neural Network based on a shallow pyramidal architecture, referred to as μPyD-Net, enabling monocular depth estimation on microcontrollers. The network is trained in a peculiar self-supervised manner leveraging proxy labels obtained through a traditional stereo algorithm. Moreover, we propose optimization strategies aimed at performing computations with quantized 8-bit data and map the high-level description of the network to low-level layers optimized for the target microcontroller architecture. Exhaustive experimental results on standard datasets and an in-depth evaluation with a device belonging to the popular Arm Cortex-M family confirm that obtaining sufficiently accurate monocular depth estimation on microcontrollers is feasible. To the best of our knowledge, our proposal is the first one enabling such remarkable achievement, paving the way for the deployment of monocular depth cues onto the tiny end-nodes of distributed sensor networks.**

***Index Terms*—Computer vision, depth estimation, deep learning, optimization methods, edge computing, IoT, micro-controllers.**

I. Introduction

Depth perception is one of the foremost cues for tackling many real-world applications like autonomous or assisted driving, robotics, safety, and security. Although for this purpose there exist effective active technologies, such as Light Detection and Ranging (LiDAR), inferring depth from images has several advantages as bulky mechanical parts are no longer needed. Therefore, it represents a long-standing problem in computer vision and different approaches, such as stereo vision and multi-view stereo, have been extensively investigated. The recent spread of machine learning (ML) has opened new frontiers and, in particular, enabled to infer depth from a single camera simplifying the setup remarkably and allowing to exploit such a cue even in applications contexts characterized by severe constraints of cost and size.

A potential downside of ML-based methods is their complexity. The Convolutional Neural Networks (CNNs) used as backbone are well known to be highly resource-demanding, both in terms of computing power and memory space, and may call for parallel accelerators like GPU cards. Nonetheless, as proposed in [1], the PyD-Net architecture leveraging an appropriate pyramidal network design enables depth perception with an accuracy comparable to state-of-the-art but with much fewer hardware requirements making it feasible on high-end CPUs commonly available in desktops, but also portable devices with a few Watts of power budget, such as flagship smartphones and tablets or wired smart cameras. As described in a more recent work [2], additional hardware-aware optimization strategies applied to the PyD-Net model, mostly aimed at reducing the data type deployed for inference, enabled to save 33% of energy with an almost equivalent degree of accuracy.

Moving along this scaling trend, it is natural to ask whether depth perception can go even further approaching smaller and cheaper off-the-shelf components able to work below the Watt mark, like tiny end-nodes powered with Micro-Controller Units (MCUs). This may offer interesting opportunities in the context of new edge applications and services in the Internet-of-Things (IoT) segment, where distributed visual sensors can evolve from simple image collectors to intelligent hubs able to infer depth locally, namely with no access to the cloud, thereby ensuring portability even in geographical areas with limited data coverage, and higher quality-of-service (QoS) thanks to more certain latency and higher users privacy [3].

Unfortunately, MCUs are orders of magnitudes less performing than embedded CPUs/GPUs, and state-of-the-art models for monocular depth estimation are too large for this purpose. Even the smallest nets, e.g. [1], [4], are designed for systems with high-speed multi-core architectures and large SRAM banks, thus consuming up to 3.5–10 W of power. Instead, typical MCUs run at a much lower frequency (hundreds of MHz vs. 1–2 GHz) and impose tight memory constraints (hundreds of kB vs. 2–8 GB). This represents a new challenge in the field of computer vision and depth perception in particular, which calls for a paradigm shift: quality is no longer the only objective, and other extra-functional metrics need to be considered at the software-level and during the whole compilation flow. Therefore, to find and implement proper

Manuscript received XX, 20XX; revised XX, 20XX; accepted XX, 20XX. Date of publication XXXX XX, 20XX; date of current version XXXX XX, 20XX. This brief was recommended by Associate Editor X. XXXX. (Corresponding author: Andrea Calimera.)

Valentino Peluso, Antonio Cipolletta and Andrea Calimera are with the Department of Control and Computer Engineering, Politecnico di Torino, Italy, 10129 (e-mail: valentino.peluso@polito.it, antonio.cipolletta@polito.it, andrea.calimera@polito.it)

Matteo Poggi, Fabio Tosi, Filippo Aleotti and Stefano Mattoccia are with the Department of Computer Science and Engineering, Università di Bologma, 40136, Italy (e-mail: m.poggi@unibo.it, fabio.tosi5@unibo.it, filippo.aleotti2@unibo.it, stefano.mattoccia@unibo.it)

resource-driven optimization strategies becomes paramount.
Pointing to this direction, the first and foremost assumption made in this work is that the processing of high-resolution images is no longer feasible, nor useful for the kind of applications addressed. It suffices to think that a single HD image may take more memory than that available on-chip. Hence, differently from the traditional high-quality vision systems, coarse depth estimates from low-resolution images is the way to meet the stringent hardware constraints of low-power MCUs, and at the same time, a strength in many edge applications concerned with privacy issues [5].
Starting from these considerations and leveraging the findings of our previous work [2], we propose:

- a novel lightweight architecture referred to as μPyD-Net specifically designed for processing low-resolution images (*i.e.* 48×48 or 32×32) on a sub-W power envelope with MCUs. The μPyD-Net maximizes the savings brought by inputs resolution scaling thanks to an internal topology optimized with hardware-conscious techniques;
- a semi-supervised training flow from low-resolution images and full-resolution disparity maps based on SGM — hence different compared to previous works [1], [2] — able to deliver enough supervision with low requirements and with equivalent performances of more costly systems.

The overall outcome of this work yields two main achievements: (i) to enable, for the first time, monocular depth estimation on low-power MCUs, such as the Arm Cortex-M7 CPU; (ii) to obtain a meaningful coarse depth representation from a tiny image, *i.e.* down to 32×32 pixels, sufficient to tackle high-level applications. We will show with extensive experimental results how despite the meager resolution of the images processed, μPyD-Net achieves, on the standard KITTI dataset [6], a depth accuracy comparable to seminal works [7], [8], although not on par with the current state-of-the-art [9]–[11]. However, this is not surprising given the much higher resolution and the thousand times more complex models of the latter methods, out of reach for MCUs used in applications at the edge.

II. Motivations and Practical Use-Cases

Despite the significant achievements in terms of accuracy, modern state-of-the-art techniques for monocular depth estimation [10], [12], [13] are overkill for many edge applications. Specifically, while high-resolution dense depth maps are desirable when dealing with tasks such as 3D reconstruction and SLAM, a rough depth estimate suffices in many applications such as object/people counting [14], [15], pose estimation [16], action recognition [5], vehicle detection [17]. Indeed, millimetric depth measurements are not strictly required in these cases to tackle the problem successfully. As an anticipation of the obtained results, a qualitative assessment is given through the use-case reported in Figure 1, which shows that the coarse maps inferred by μPyD-Net can be used for vehicle detection seamlessly, by simply looking for differences in the coarse 3D structure of the scene with respect to the structure of environment itself in absence of vehicles. Intuitively, with lighter input images the hardware requirements reduce substantially enabling to bring the task within the computing capability of MCUs. Obviously, the limited operating frequency and low parallelism of MCUs prevent real-time performance (*i.e.* $>$30 FPS). However, the focus here is on those applications with relaxed timing constraints, like traffic congestion monitoring, and not fast decision making needed, for instance, on autonomous driving. However, in the case of decision-making systems, our solution can be easily ported to mobile CPUs in order to gain about $100\times$ performance at the cost of only $10\times$ power consumption.
Moreover, in many circumstances, processing low-resolution images is even desirable. For instance, depth predictions can be used as privacy-preserving features for further processing in cloud systems. Commonly referred to as *collaborative intelligence* [19], this strategy enables to preserve user privacy, masking sensitive data that are present in raw RGB images. An example is depicted in Figure 2, which applies to an image extracted from the VAP dataset [18]. Distinctive features in data managed by the staff in charge of monitoring the environment may violate users' privacy. Although low-resolution images partially help to alleviate this issue, sufficiently distinctive clues can still be inferred as we can notice from Figure 2 c). However, by moving to a pure depth domain as in Figure 2 d), one can hide details yet keeping the relevant information required for the high-level task. Bringing these sensing capabilities on tiny devices is paramount to reduce design costs and power consumption of systems based on RGB-D cameras (e.g. Kinect in (b)) or standard network for monocular depth estimation (d). The map inferred by μPyD-Net (e) and post-processed by a super-resolution network running on the cloud (f) achieves results comparable to the two baselines.
To play with low-resolution inputs is beneficial in terms of the number of operations to be processed but also memory footprint to store hidden features maps. However, an in-depth analysis conducted in Sec. VII-D demonstrates that resolution scaling alone is not enough to fit current models on MCUs and that to achieve the goal requires much more design and optimization efforts. This motivates the need of novel inference models, like the proposed μPyD-Net which, thanks to its compact topology, a novel training procedure, and the optimization pipeline, ensures prediction quality not far from that of standard techniques processing high-resolution images.

III. Related Work

In this section, we review the literature concerning monocular depth estimation and deep networks compression since both topics are pertinent to our work.

A. Depth Estimation

Inferring depth from images, *i.e.* obtaining the distance of real-world points framed by a camera, is a long-standing problem in computer vision, and it can be exploited to tackle a plethora of problems as environment monitoring, object detection, collision avoidance, and many more. Although *active* sensor, such as *Time-of-Flight* or *LiDAR*, can be used for this purpose,

Fig. 1. Example of application concerning traffic monitoring with minimum memory requirements and power consumption enabled by μPyD-Net. For each example, we show the high-resolution frame, followed by the 32×32 image processed by the network (top-right corner) and its output (mid-right corner). In the bottom-right corner, for each example, we show the differences in the coarse 3D structure of the scene with respect to the structure of the environment itself, which has been acquired in absence of vehicles (top-left example). The highlighted regions depict the changes in the depth maps inferred by μPyD-Net when a vehicle enters the scene.

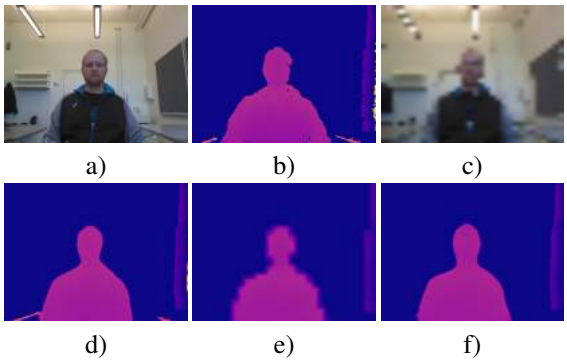

Fig. 2. Results on a testing image of the VAP dataset [18]. a) RGB original frame, b) ground-truth depth acquired using Kinect, c) RGB input image resized to 32×32, d) maps predicted by PyD-Net, e) maps predicted by μPyD-Net, and f) outcome of the super-resolution network fed with the μPyD-Net map.

passive ones generally have some notable advantages. Specifically, they do not perturb the sensed environment, rely on fast and inexpensive imaging devices with an ever-increasing resolution, require neither any additional hardware equipment nor have any moving mechanical parts.

Given a stereo rig, *i.e.* a setup with two aligned and synchronized cameras, the geometrical (*epipolar*) constraint [20] allows to infer depth by mean of *triangulation* once the correspondence between pixels in the two views is addressed. Due to its central relevance in computer vision, since the '60s many algorithms have been proposed in the literature [21]–[24]. Recently, thanks to the more and more increasing capabilities of hardware devices, *deep learning* proved to be able to tackle the complexity of such a problem allowing to reach astonishing results. In particular, starting with [25] many approaches have been proposed [26]–[29]. Although stereo vision represents a prevalent solution concerning depth from images, it requires two calibrated cameras and synchronized image acquisition. Thus, this setup may limit its practical deployment when it is not available or feasible (*e.g.* on existing installations), too expensive or cumbersome. Moreover, it becomes unreliable in the case of miscalibration or when large dis-occlusions occur between the two cameras due to the particular image content framed.

Therefore, to prevent all previous issues, a recent trend in this field consists in inferring depth using a single camera. Despite monocular depth estimation is a strongly *ill-posed* problem, in the last few years, deep learning enabled to obtain compelling results. Specifically, [7], [8], [10]–[13], [30]–[39] adopted different strategies to obtain reliable depth estimation from a single camera by learning to exploit monocular clues such as shadows, occlusions and relative scales between objects. In this field, a particularly appealing practice consists of training *end-to-end* models in a self or semi-supervised manner [12], [36], replacing the need for ground-truth depth labels with image reprojection across different viewpoints according to two main strategies, respectively acquiring images with a single, moving camera [36], [38], [40], [41] or using a stereo camera [10], [12], [37], [42]–[44]. The latter approach usually turns out more effective, in particular when deploying self-supervision extracting proxy labels using stereo algorithms [10], [11]. Unfortunately, despite their effectiveness, these architectures are quite complex and for inference generally require high computing capabilities (*e.g.* GPUs) and have large memory footprints. Consequently, they also lead to high power consumption, frequently around 250 W. For these reasons, they are not suited at all for embedded devices. Nonetheless, this issue has been partially addressed in recent works [1], [2] aimed at obtaining real-time performance, possibly on embedded devices, paying a low or negligible contribution in terms of accuracy. Despite these remarkable achievements, monocular depth estimation on low-power MCUs has never been addressed before.

B. Quantization

Quantization is a reduction technique commonly used to improve the processing of CNNs. Its adoption is paramount for deploying CNNs into devices with tight memory constraints and limited computational resources, like low-power MCUs. Let's consider the Arm Cortex-M family for instance, widely used in low-cost and power-efficient portable systems. First, the flash memory is limited to few MBs, while the SRAM size ranges from tens to hundreds of KBs only. The flash memory

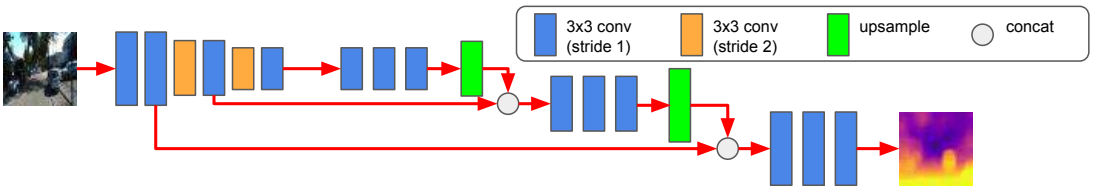

Fig. 3. μPyD-Net architecture. Three levels of features are extracted by the pyramidal encoder, while three compact decoders gradually restore the original input resolution and produce depth predictions. Overall, μPyD-Net counts only about 100K parameters.

stores the binary of the application, including the network weights, which is block-loaded into the SRAM at run-time to ensure faster execution. The SRAM also stores intermediate features of the network, whose size is not negligible in deep CNNs like μPyD-Net. Intuitively, 8-bit quantization is highly desirable as it enables a $4\times$ reduction of the memory footprint with respect to 32-bit, still with negligible accuracy degradation. Second, most of the low-power RISC-based architectures have a limited instruction set. In the Cortex-M architectures, the floating-point units are optional for many chip-sets, and even if available there is no support for vector instructions. To meet the throughput requirements of typical computer vision applications, the only practical option is to exploit the parallelism of the 2-way Single-Instruction Multiple-Data (SIMD) integer data-path (available on the M7 core). This suggests quantization becomes a must rather than an optimization option.

The aim of this subsection is to briefly summarize the key aspects of quantization and to provide a synthetic review of existing techniques. Earlier works showed that a 32-bit floating-point representation is redundant at inference time. The most of existing CNNs can be quantized to 16-bit and 8-bit integers [45] with no, or minimal loss on the output quality. More recent works also proposed extreme bit-width lowering, down to ternary [46] or binary [47] representation, yet with a substantial quality drop depending on the application and the network topology. However, the lowest integer option is the 8-bit for off-the-shelf MCUs. More scaled precision can be only implemented by custom software routines that induce substantial performance overhead [48]. In view of the above, 8-bit quantization represents the best trade-off between memory compression and performance.

Floating-point numbers can be discretized using different schemes investigated in the past in the field of Digital Signal Processing (DSP). They can be broadly classified as *linear* and *non-linear*. According to the linear scheme, the quantization step, *i.e.* the distance between two adjacent integer values, is kept constant across the entire input range. This allows a straightforward implementation that is based on simple arithmetic operations. The mapping can be *symmetric*, if the integer distribution is centered around zero, or *asymmetric* if shifted by a given offset; the first choice is simpler, while the second one reaches the best fit on the original floating-point distribution with an additional cost due to the offset. Finally, it is possible to adopt a *power-of-two* scaling or an *arbitrary* scaling factor. The former makes use of simple shift operations for scaling among the quantized levels [49], the latter might be more accurate but it generally requires additional arithmetic [49]. Concerning the non-linear quantization, it applies a custom function for encoding the original real data on a discrete set of integer values. Common approaches are *logarithmic* quantization [50] or *clustering* [51]. They achieve the highest quality when the distribution of the original data is not uniform. Obviously, the implementation introduces additional overhead, mainly due to custom procedures to be stored and run. As a rule of thumb, accurate strategies, *i.e.* linear/asymmetric and non-linear quantization, achieve higher fidelity at the cost of lower performance [49]. For this specific reason, we adopted a linear quantization scheme with a power-of-two scaling, more suited for general purpose low-power MCUs. As it will be discussed in Section VII-D, the strategy adopted in this work guarantees accuracy comparable to a floating-point representation. Therefore, higher efficiency comes with no penalty on the quality-of-results.

C. Pruning

Pruning is a reduction technique orthogonal to quantization. Concerning compression pipelines for deployment onto MCUs, a joint application of the two strategies is studied in [52] for classification tasks.

Based on the assumption that CNNs are over-parametrized, pruning strategies aim to identify and remove those parameters with a negligible contribution to the predictive quality of the model. The key difference among the existing implementations lies in the level of spatial granularity at which pruning operates. *Weight-pruning* [51] is the finest level of granularity, *i.e.* every single weight can be removed. To keep a regular shape of the weight matrices, unimportant parameters are simply zeroed. This increases the overall sparsity, *i.e.* the ratio between zero and non-zero parameters, thereby enabling compression methods based on sparse data representations such as Huffman coding. *Group-pruning* represents a middle level of granularity. Weights are pruned in blocks of a size such that the utilization of the parallel arithmetic units of the hosting hardware is maximized [53]. *Filter-pruning* [54] is the coarsest level. It works in a structured manner, namely entire convolutional filters are dropped reducing both memory footprint and number of operations.

In the context of monocular depth estimation, a recent study [55] demonstrated that pruning can reduce the number of parameters up to $5\times$ with competitive accuracy. However, porting standard monocular depth estimation models on MCUs requires a much more aggressive compression, *i.e.* $>100\times$, as will be shown later in the text (Section VII-C). Motivated

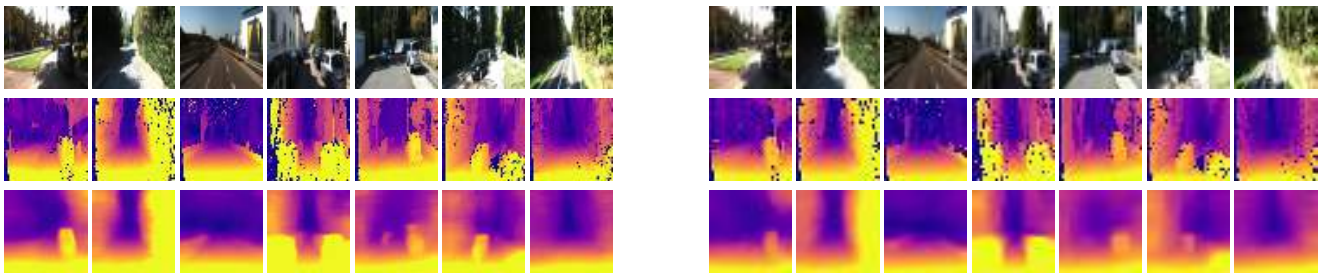

Fig. 4. Examples of self-sourced proxy labels on 48×48 (left) and 32×32 (right) images. From top to bottom, we show reference images, disparity maps produced by SGM [22] (on full resolution images, then downsampled by means of nearest neighbor interpolation) and predictions by μPyD-Net. In the SGM map, outliers detected by the left-right consistency check depicted in dark blue.

by this observation, we resorted to topology restructuring in order to obtain an already compact model that fits the available memory budget.

IV. μPyD-Net architecture

In order to accomplish monocular depth estimation under the challenging constraints outlined, two main factors need to be carefully taken into account to keep both manageable memory footprint and execution time: input spatial resolution and network complexity, with the former usually driving most of the design choices linked to the latter. For instance, pooling and stride parameters in convolution are adjusted to enlarge the receptive field as well as to reduce the computational burden at higher resolutions. Thus, the first design choice to meet our constraints consists of inferring inverse depth from a small input image resulting in an extremely compact model, namely *micro*PyD-Net (μPyD-Net). Figure 3 sketches our architecture. Following the PyD-Net design, a shallow encoder extracts a three-level pyramid of features using six 3×3 convolutional layers followed by leaky ReLU activations having $\alpha = 0.125$, producing respectively 8, 8, 16, 16, 32, and 32 features. According to Figure 3, orange layers apply a stride factor of 2, halving the spatial resolution. Then, three decoders made of three convolutional layers, followed by leaky ReLU (except the last one), process each level of the pyramid producing 32 features each. The output of the last layer is up-sampled through a 2×2 transposed convolution layer. The extremely compact architecture, counting barely 100K parameters, is thought to run on tiny resolution images and thus is tailored to low power devices such as MCUs. This, coupled with appropriate image resolutions, *i.e.* 48×48 and 32×32, allows for breaking the 512 kB memory and 1 FPS barriers on such a low-powered device, as we will show in detail in our evaluation. Adding more layers either to the features extractor or the decoders would make one or both the requirements not met. Moreover, in our experiments, we will prove how processing 48×48 and 32×32 will allow for deployment on such family of devices and still source results accurate enough for several high-level applications. In addition to the issues induced by processing low-resolution images, such as loss of details, providing supervision at such resolution becomes challenging. In particular, when the annotation is sparse like in the KITTI dataset [6], downsampling such sparse depth data to the small input resolution of our network would make labels no longer reliable because of interpolation. For this reason, during training, we rely on proxy-supervision [10] deploying a traditional stereo algorithm such as Semi-Global Matching (SGM) [22].

V. Proxy-supervision

Since sourcing accurate depth labels is expensive and time-consuming, several works replaced the need for accurate ground truth labels using view synthesis [12], [36] deploying pairs of synchronized images acquired by a stereo camera to exploit a re-projection loss for supervision [12]. That is, given a stereo pair made of images L and R, the network is trained to infer from L an inverse depth map (*i.e.* disparity) D^L. Then, we warp R according to D^L so as to obtain $\tilde{R}$. A photometric loss $\mathcal{L}_{ap}^{l}$ between L and warped image $\tilde{R}$ defined as in [12]

$$\mathcal{L}_{ap}^{l} = 0.85 \cdot \frac{(1 - \text{SSIM}(L, \tilde{R})}{2} + 0.15 \cdot |L - \tilde{R}| \quad (1)$$

supervises the network. The network can be trained also to infer a synthetic disparity map D^R for the right image R so as to enforce consistency between the two. In this case, an equivalent $\mathcal{L}_{ap}^{r}$ signal can be obtained comparing R with warped image $\tilde{L}$. In our previous work [2] we followed this strategy. Nevertheless, we argue that at such a low resolution, the photometric loss alone is not enough to obtain sufficiently reliable supervision. Hence, we follow a different strategy.

In this field, a further step forward consists of using traditional stereo algorithms [21] to produce noisy disparity estimations and leverage on them for supervision. This latter strategy proved to be very effective for self-supervised training of both stereo [56], [57] and monocular [10], [57] networks, outperforming re-projection losses.

We follow this strategy also to avoid downsampling of sparse ground truth labels, which would result in even sparser annotations or incorrect values introduced by interpolation when the depth data is not available at the reduced resolution. Purposely, as in [10], we use the SGM algorithm [22] to generate dense proxy labels from stereo pairs. For each pixel p and disparity hypothesis d, a Hamming matching cost $C(p, d)$ is computed between 9×7 census transformed images, then it is refined according to multiple scanline optimizations as follows:

$$\begin{aligned} E(p,d) = & C(p,d) + \min_{q>1}[C(p',d), C(p',d \pm 1) + P1, \\ & C(p',d \pm q) + P2] - \min_{k<D_{max}} (C(p',k)) \end{aligned} \quad (2)$$

with P_1 and P_2 two smoothness penalties, discouraging disparity gaps between p and previous pixel p' along the scanline

path. A *winner-takes-all* strategy is applied after summing up the outcome of each optimization phase. Finally, the *left-right consistency* constraint [21] is enforced to filter out outliers as follows. By computing disparity maps D^L and D^R with SGM, respectively assuming as reference left and right images, we invalidate pixels having different disparities across the two maps:

$$D(p) = \begin{cases} \tilde{d}(p) & \text{if } |D^L(p) - D^R(p - D^L(p))| \leq \varepsilon \\ -1 & \text{otherwise} \end{cases} \tag{3}$$

As reported in the remainder, to validate our approach, we rely on the popular high-resolution KITTI dataset [6]. Hence, in order to obtain proxy labels as accurate as possible, we run SGM on images at the original resolution $W \times H$, then we downsample them respectively to 48×48 and 32×32 using nearest neighbor interpolation, and opportunely scale disparity values by $\frac{48}{W}$ and $\frac{32}{W}$, to effectively obtain proxy labels at the same resolution of the network inputs. It is worth to note that such labels are not entirely dense since outliers are filtered-out enforcing the left-right consistency constraint. Nonetheless, most points survive this process, and each valid value available in the inverse depth map is obtained without any interpolation from nearby points.

The obtained labels are then used to provide supervision to μPyD-Net employing a reverse Huber (berHu) loss [58]

$$\mathcal{L}_{ps} = \frac{1}{N} \sum_{p} berHu(d(p), \tilde{d}(p), c) \tag{4}$$

$$berHu(d(p), \tilde{d}(p), c) = \begin{cases} |d(p) - \tilde{d}(p)| & \text{if } |d(p) - \tilde{d}(p)| \leq c \\ \frac{|d(p) - \tilde{d}(p)|^2 - c^2}{2c} & \text{otherwise} \end{cases} \tag{5}$$

where $d(p)$ and $\tilde{d}(p)$ are, respectively, the predicted disparity and the proxy annotation for pixel p while c is set as $\alpha \max_p |d(p) - \tilde{d}(p)|$, with $\alpha = 0.2$.

Figure 4 shows, from top to bottom, some qualitative examples of low-resolution images (48×48), followed by proxy labels generated by SGM and disparity maps estimated by μPyD-Net. One can notice how the network accurately reproduces inverse depth estimations consistent with the self-sourced annotations. Finally, as in [10], we sum to proxy-supervision the contribution given by photometric loss to obtain our final loss as

$$\mathcal{L}_{init} = \alpha_{ap}(\mathcal{L}^l_{ap} + \mathcal{L}^r_{ap}) + \alpha_{ps}(\mathcal{L}^l_{ps} + \mathcal{L}^r_{ps}) \tag{6}$$

We tuned α_{ap} and α_{ps} following [10]. Although SGM is effective, additional sources of proxy labels can be stereo networks trained in a self-supervised manner with photometric losses, as shown in [59], or in a supervised manner at the cost of requiring ground truth labels. In our ablation experiments, we will thoroughly study the impact of the different strategies.

VI. Optimization Stack

The optimization stack designed for the deployment of μPyD-Net into Cortex-M MCUs is depicted in Figure 5. It consists of two main stages: (i) the *front-end*, where the model is quantized using 8-bit fixed-point representation; (ii) the *back-end*, where the high-level description of the quantized network is translated into low-level routines optimized for the target device.

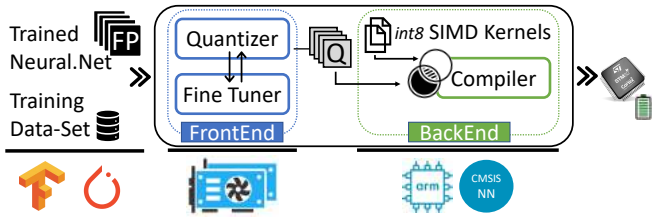

Fig. 5. Pictorial representation of the optimization framework used to translate the high-level description of a neural network in an optimized version ready to deployed on a MCU powered by a Cortex-M CPU.

As previously introduced, the quantization is built following a linear scheme with power-of-two scaling in order to efficiently exploit the integer data-path of the Cortex-M architecture. More in detail, we adopted a dynamic approach by which the radix-point of both feature maps and weights is assigned layer-by-layer. To calculate the optimal radix-point, we developed a simple heuristic that returns the optimal fraction length of each layer such that the mean squared error between the original floating-point distribution and the quantized one is minimized. For intermediate features, we collected the statistics on a subset of the training set (referred to as the calibration set). The accuracy loss due to quantization is then recovered through a *fine-tuning* stage based on knowledge distillation [60]. The quantized model, set as the student, is re-trained to *mimic* the original floating-point network, the teacher. As a training loss, we adopted the mean squared error between the disparity maps inferred by the two actors (teacher and student). An important aspect to be noticed is that the re-training of the quantized model encompasses the execution of the integer model. Since GPUs do not support integer arithmetic (at least those available in our setup) we implemented an emulation framework, built upon the concept of *fake-quantization* [61] and tuned to be compliant with the arithmetic units of the Cortex-M cores. It is a software wrapper that converts activations and weights (stored in fixed-point) to the 32-bit floating-point; after being processed, results are converted back to fixed-point. We experimentally tested the flow, and the results produced by emulation exactly match those collected on the target hardware.

After quantization and fine-tuning, the network is ready to be deployed on the target device. The porting stages leverage the CMSIS-NN library developed by Arm. It is a collection of handwritten routines that ensure efficient processing of integer CNNs. Unfortunately, the CMSIS-NN was mainly designed for simple tasks, like image classification and keyword-spotting [49], hence it supports a limited set of operators. We augmented the library with optimized routines for the missing operators: deconvolution and leaky ReLU. For deconvolution, the input features are upsampled using a factor equal to the stride, then convolved with unit stride. For the leaky ReLU, the slope is constrained to be a power-of-two, hence it can be implemented with a simple shift operation. We observed that this choice achieves better performance with no impact on the

final accuracy.

VII. Experimental Results

In this section, we describe the datasets, the implementation details, and report exhaustive experiments aimed at assessing the performance of μPyD-Net, according to both functional (*i.e.*, accuracy) and extra-functional (*i.e.*, latency and memory footprint) metrics; to notice that the energy consumption is inversely proportional to latency. This two-fold evaluation will show how μPyD-Net performs in terms of depth accuracy compared to much more complex state-of-the-art solutions, highlighting its much superior efficiency and suitability for low-power MCU platforms.

A. Dataset & Training

Our quantitative evaluation primarily involves two datasets: KITTI [6] and CityScapes [62]. Moreover, we use the Make3D dataset [30] for additional experiments concerning the generalization of μPyD-Net and other state-of-the-art models.

KITTI. The KITTI stereo dataset [6] is a collection of rectified stereo pairs made up of 61 scenes (more than 42K stereo frames) concerned with driving scenarios and it is the standard dataset for evaluating monocular depth estimation methods as well as for many other purposes. The average image resolution is 1242×375 and a LiDAR device, mounted and calibrated in proximity to the left camera, was deployed to measure depth information. Following other works in this field [7], [12], we divided the overall dataset into two subsets, composed respectively of 29 and 32 scenes. This subdivision is referred to as *Eigen split*. We used 697 frames belonging to the first group for testing purposes and 22600 more taken from the second for training.

CityScapes. The CityScapes dataset [62] contains stereo pairs concerning about 50 cities in Germany taken from a moving vehicle in various weather conditions. It consists of 22,973 stereo pairs with a resolution of 2048×1024 pixels. It is often used for pre-training [10], [12], [37] before moving to the KITTI dataset, but not for evaluation since no ground truth maps are provided (only disparity maps computed with SGM). As in [12] the lowest 20% of each stereo pair is discarded at training time.

Make3D. The Make3D dataset [30] consists of a set of images and depth maps from a custom-built 3D scanner, collected during daytime in a diverse set of urban and natural areas in the city of Palo Alto and its surrounding regions. It contains 534 images at 1704×2272 resolution. We run experiments on the 134 testing images without retraining as in [12], [63].

B. Hardware Set-up

The proposed μPyD-Net is tested and validated on a NUCLEO-F767ZI [66] development board manufactured by ST-Microelectronics. It hosts a chip-set powered with an Arm Cortex-M7 CPU with $216\,\mathrm{MHz}$ clock frequency, $512\,\mathrm{kB}$ of SRAM and $2\,\mathrm{MB}$ of flash memory. As reported in the datasheet [67], the current consumption is $\approx 100\,\mathrm{mA}$ for a data-intensive application run under the same operating condition of our experiments, and hence a resulting power consumption $< 400\,\mathrm{mW}$. The `.C` description of the optimized μPyD-Net model is built using the GNU Arm Embedded Toolchain, version 6.3.1, and flashed into the board using the mbed-cli toolchain.

The optimization framework is run on a workstation powered by a GPU NVIDIA GTX-1080 Ti. Extensive simulations on the KITTI dataset validated the integer emulation engine integrated into the front-end side. Collected traces show 100% accuracy with respect to on-board measurement.

TABLE I
PROXY LABELS ACCURACY ON THE TEST SET OF KITTI DATASET [6] USING THE SPLIT OF EIGEN ET AL. [7], MAXIMUM DEPTH SET TO 80M.

		Lower is better ⇓				Higher is better ⇑		
Method	Resolution	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
SGM [22]	native	0.064	0.584	3.700	**0.149**	**0.951**	**0.976**	**0.986**
UnOS [64]	native	0.064	0.582	3.690	0.169	0.932	0.964	0.979
DispNet-CSS [65]	native	**0.060**	**0.442**	**3.543**	0.167	0.940	0.967	0.981
SGM [22]	48×48	0.415	10.963	11.836	0.481	0.539	0.785	0.881
SGM [22]	$\downarrow 48 \times 48$	0.107	1.594	5.556	0.199	0.906	0.960	0.978
UnOS [64]	$\downarrow 48 \times 48$	0.102	1.145	5.140	0.197	0.901	0.955	0.976
DispNet-CSS [65]	$\downarrow 48 \times 48$	**0.089**	**0.723**	**4.420**	**0.183**	**0.909**	**0.961**	**0.980**
SGM [22]	32×32	0.652	21.745	15.319	0.638	0.370	0.667	0.797
SGM [22]	$\downarrow 32 \times 32$	0.133	2.083	6.448	0.222	0.873	0.949	0.974
UnOS [64]	$\downarrow 32 \times 32$	0.131	1.622	6.040	0.221	0.867	0.945	0.972
DispNet-CSS [65]	$\downarrow 32 \times 32$	**0.110**	**0.940**	**4.948**	**0.199**	**0.882**	**0.955**	**0.978**

TABLE II
ABLATION STUDY ON THE TEST SET OF KITTI DATASET [6] USING THE SPLIT OF EIGEN ET AL. [7], MAXIMUM DEPTH SET TO 80M.

		Lower is better ⇓				Higher is better ⇑		
Supervision	Res.	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Photo	48×48	0.260	3.386	9.416	0.391	0.593	0.805	0.903
SGM	$\downarrow 48 \times 48$	0.200	2.107	7.144	0.295	0.707	0.871	0.943
UnOS	$\downarrow 48 \times 48$	0.200	2.095	7.224	0.298	0.701	0.870	0.944
DispNet-CSS	$\downarrow 48 \times 48$	**0.191**	**1.825**	**6.697**	**0.286**	**0.716**	**0.881**	**0.949**
Photo	32×32	0.315	4.984	11.007	0.451	0.539	0.764	0.879
SGM	$\downarrow 32 \times 32$	0.221	2.547	7.625	**0.312**	**0.681**	**0.858**	**0.935**
UnOS	$\downarrow 32 \times 32$	0.221	2.625	7.623	0.313	0.677	0.853	**0.935**
DispNet-CSS	$\downarrow 32 \times 32$	**0.217**	**2.195**	**7.171**	0.314	0.680	0.855	0.934

C. Evaluation – Functional metrics

We evaluate predictions according to standard functional metrics [7], [12]: *Abs rel*, *Sq rel*, *RMSE* and *RMSE log* represent error measures (⇓, the lower the better), while $\delta < K$ the percentage of predictions whose maximum between ratio and inverse ratio with respect to the ground truth is lower than a threshold K (⇑, the higher the better). The detailed formulation of each metric can be found in [7].

Proxy labels evaluation. Since the performance of μPyD-Net is limited to the accuracy of the proxy labels used for supervision, we study different strategies to source such data and how they behave at low resolution. In particular, although SGM represents a popular choice and trade-off in terms of accuracy and speed, more accurate methods exist. To this aim, we consider labels obtained by means of SGM and by distillation [59] from two state-of-the-art neural networks, respectively trained with self-supervision and ground truth. Specifically, we choose UnOS by Wang *et al.* [64], currently state-of-the-art for self-supervised stereo, and DispNet-CSS by Ilg *et al.* [65]. For both we use the weights made available by the authors, respectively trained with the photometric loss on the full KITTI dataset (UnOS) or with ground truth on the SceneFlow dataset [26] and fine-tuned on KITTI 2015

TABLE III
QUANTITATIVE EVALUATION ON THE TEST SET OF KITTI DATASET [6] USING THE SPLIT OF EIGEN ET AL. [7] WITH MAXIMUM DEPTH SET TO 80M. METHODS WITH * RUN POST-PROCESSING [12].

				Lower is better ⇓				Higher is better ⇑		
Method	Resolution	Params	MCU	Abs Rel	Sq Rel	RMSE	RMSE log	δ <1.25	$\delta < 1.25^2$	$\delta < 1.25^3$
Eigen et al. [7] Fine	172×576	54.2M	No	0.203	1.548	6.307	0.282	0.702	0.890	0.958
Liu et al. [8]	-	40.0M	No	0.201	1.584	6.471	0.273	0.680	0.898	0.967
Zhou et al. [36]	128×416	34.2M	No	0.198	1.836	6.565	0.275	0.718	0.901	0.960
MonoDepth [12] ResNet50*	256×512	48.0M	No	0.114	0.898	4.935	0.206	0.861	0.949	0.976
3Net [37] ResNet50*	256×512	65.0M	No	0.111	0.849	4.822	0.202	0.865	0.952	0.978
MonoDepth2 (S) [63]	192×640	11.0M	No	0.109	0.873	4.960	0.209	0.864	0.948	0.975
DSVO [9]*	256×512	96.2M	No	0.097	0.734	4.442	0.187	0.888	0.958	0.980
MonoResMatch [10]*	384×1280	42.5M	No	**0.096**	**0.673**	**4.351**	**0.184**	**0.890**	0.961	**0.981**
DepthHints [11]*	320×1024	34.5M	No	**0.096**	0.710	4.393	0.185	**0.890**	**0.962**	**0.981**
PyD-Net [1]	256×512	1.9M	No	0.146	1.291	5.907	0.245	0.801	0.926	0.967
μPyD-Net	48×48	0.1M	Yes	0.193	2.312	6.952	0.277	0.735	0.890	0.953
μPyD-Net	32×32	0.1M	Yes	0.215	2.395	7.252	0.301	0.696	0.866	0.939

training set (DispNet-CSS). We point out that, being ground truth labels required to train DispNet-CSS, distilling proxy labels through this model adds some constraints to such a solution.

Table I collects results concerning a comparison between the three, conducted on the Eigen test split. We first report the accuracy of the disparity maps processed at full resolution, to highlight the importance of spatial resolution. We highlight the almost equivalent performance by SGM and UnOS on error metrics, with DispNet-CSS producing better results. Concerning deltas, SGM produces better accuracy. Considering the 48×48 resolution, we report four main experiments respectively evaluating disparity map obtained by running SGM on 48×48 images and by either SGM, UnOS, and DispNet-CSS at full resolution and then downsampled by means of nearest-neighbor interpolation ($\downarrow 48 \times 48$), *i.e.* to the resolution used to train μPyD-Net. We cannot run either UnOS or DispNet-CSS at 48×48 because of their high compression factor ($\frac{1}{64}$), requiring larger images. We point out the extremely bad performance achieved by running SGM on 48×48 images, because of the high quantization of pixels at this resolution, making this solution unreliable both for applications on microcontrollers as well as for training neural networks to run on these latter. Conversely, full-resolution images downsampled to 48×48 maintain acceptable performance, with DispNet-CSS labels resulting much more accurate. In general, SGM and UnOS are close in performance, with the former resulting slightly more accurate and thus preferable to train μPyD-Net. The same behavior can be observed by running experiments at 32×32 resolution. Although DispNet-CSS labels show much higher accuracy compared to SGM and UnOS, they need ground truth labels to be obtained. We will highlight next how training μPyD-Net on DispNet-CSS rather than SGM achieves only minor improvements, thus making SGM better suited for practical applications.

Ablation study on μPyD-Net. Finally, we study the effectiveness of μPyD-Net variants trained with different sources of self-supervision. Table II collects results of 48×48 and 32×32 models trained respectively with image reprojection losses [12], proxy labels sourced through SGM algorithm, UnOS and DispNet-CSS. At first, we point out how the supervision from photometric losses performs much worse compared to the use of proxy labels. Although this approach is extremely popular [1], [12], [37], we argue that, intuitively, the image content at such low resolution is much lower compared to the one available at the original resolution, thus leading to poor supervision. Exploiting the guidance from accurate disparity maps at training time allows to greatly boost the accuracy achieved by μPyD-Net. Nevertheless, although the proxy labels show different accuracy according to Table I, in particular comparing row 4, 5,6 and 8, 9, 10 sourcing supervision from SGM algorithm results slightly better than using UnOS, with a minor margin compared to DispNet-CSS, although DispNet-CSS needs ground truth for training conversely to SGM. Thus, for practical applications, we prefer the SGM solution because of the low margin with respect to DispNet-CSS moderate, yet the much greater flexibility.

Comparison with state-of-the-art. In Table III we compare μPyD-Net with state-of-the-art solutions for monocular depth estimation. The upper portion of the table contains complex architectures with millions of trainable parameters, suited only for high-end GPUs (*e.g.* the NVIDIA Titan XP). On the other hand, the lower portion of the table lists networks requiring much less computational and memory requirements compatible with a broader range of devices. Moreover, we also report for each network the resolution of the input image processed. At first glance, we can notice the large gap between the amount of information processed by μPyD-Net and other proposals. However, shrinking the image using this extreme factor (down to $\frac{1}{38}$ for width, $\frac{1}{11}$ for height in case of 32×32 images) has a non-negligible impact on the input image fed to μPyD-Net. This degradation is particularly evident for small objects at a longer distance or thin structures as poles, causing higher errors compared to the ground-truths acquired at full-resolution. For this reason, Table III also includes the lightweight PyD-Net [1] network processing much larger 256×512 images. Nonetheless, it is important to notice that even stretching the input of other proposals to either 48×48 or 32×32 they would not be able to run on the targets hardware device due to excessive memory requirements. Moreover, PyD-Net [1] would not be compatible with such tiny image sizes since its pyramidal structure is too deep. However, as pointed out by previous studies in other fields [68], [69], the image content encoded in such tiny images is still enough

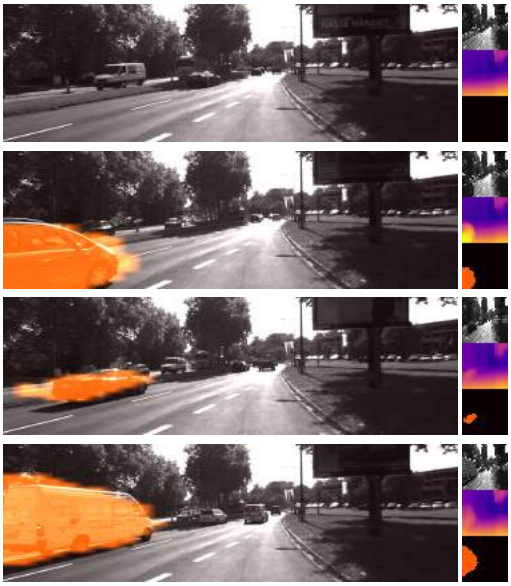

Fig. 6. Qualitative results concerning traffic monitoring. For each example, we show the high-resolution frame, followed by 32×32 images processed by μPyD-Net.

to estimate a coarse estimation of the scene, comparable to state-of-the-art techniques proposed just a few years ago [7], [8], with hundred times fewer parameters and computational requirements. Not surprisingly, 48×48 input images yield better results compared to 32×32.

Comparison with state-of-the-art models on low-resolution images. To further support that μPyD-Net is effective at extracting most of the knowledge available from low-resolution images, we show the performance achieved by two state-of-the-art networks, respectively MonoDepth2 [63] and MonoResMatch [10], when processing 32×32 images. Being these latter not able to process such tiny images because of architectural limitations, we simulate low-resolution images by downsampling the inputs to 32×32 ($\downarrow 32 \times 32$) and then upsampling ($\uparrow$) them back to the original resolution. Table IV collects the outcome of this evaluation, showing how μPyD-Net places in between the two competitors for most metrics (*i.e.*, Sq Rel, RMSE, RMSE log, $\delta < 1.25$, $\delta < 1.25^3$) although counting two order of magnitude less parameters. This supports the fact that μPyD-Net itself is enough to extract most of the information available from low-resolution content, while keeping low complexity. This latter property is crucial for deployment on the target microcontrollers, over which MonoDepth2 and MonoResMatch parameters alone would not fit into the available memory.

Close-range and quantization evaluation. We stress the fact that the farther points in the scene are those most affected by the degradation introduced by processing tiny images since each pixel senses a larger portion of the real scene. Therefore, we will assess the accuracy of μPyD-Net when sensing at different ranges the scenes included in the datasets. Table V reports a detailed comparison between μPyD-Net and its optimized counterpart considering different depth ranges, from $0\,\mathrm{m}$ to 15, 25, 50 and $80\,\mathrm{m}$. In the upper portion results obtained by μPyD-Net processing 32×32 images and in the middle processing with the same network 48×48 images. On the very bottom of the same table, we also report for comparison results yielded by state-of-the-art [10]. We can notice in general how, independently of the input resolution and evaluation range, introducing the quantization it dramatically drops the performance of μPyD-Net (float32 vs int8 entries), as already observed in [2]. However, by fine-tuning the model after quantization (int8-ft entries), the original performance is restored for most metrics and sometimes even improved. Focusing on how the metrics change across the different evaluation ranges, we can perceive how on nearby measurements the gap between μPyD-Net and much more complex state-of-the-art [10] gets lower. For instance, by looking at the RMSE metric, we can observe how the difference in terms of average error is about 3 when considering the full evaluation range 0-$80\,\mathrm{m}$, while it drops to about 0.6 and 0.8 respectively for 48×48 and 32×32 images when dealing with the 0-$15\,\mathrm{m}$ range. This behavior suggests that μPyD-Net might be not particularly suited for long-range depth measurements. However, for close-range depth sensing, it provides a valid alternative when a low-power budget is paramount. For instance, Figure 6 shows a qualitative example of a traffic monitoring system processing images from the KITTI dataset [6] downsampled to 48×48 resolution. We show four images acquired from a static point of view to simulate a monitoring camera placed on a crossroad. For each one, we report on the right their downsampled counterpart, the estimated disparity map sourced by μPyD-Net and a segmentation map detecting objects on the scene over imposed to the original KITTI image. To this aim, given the depth layout estimated for the empty scene (*i.e.*, in absence of vehicles) estimated by μPyD-Net, a simple change-detection algorithm in the depth domain is sufficient to detect nearby cars reliably. Although this application allows for simple traffic monitoring, the depth cue provided by μPyD-Net can be exploited for other purposes (*e.g.*, 3D tracking) and to replace other sensors as well. Therefore, such information could be used in place of other sensors or to enrich other image-based cues such as object detection or semantic segmentation.

Generalization on Make3D dataset. In order to assess the generalization properties, in Table VI we report results on the Make3D dataset [30] following the evaluation proposed in [63], on a center crop of 2×1 ration and without applying median scaling (not required when training on stereo pairs). We point out how state-of-the-art networks suffer from huge drops when moved to unseen environments as well. μPyD-Net can still provide meaningful predictions, very close to those by MonoDepth when running at 48×48. Figure 7 show some qualitative examples, showing in particular how the coarse disparity maps by μPyD-Net are often less affected by artifacts with respect to the predictions by MonoResMatch.

D. Evaluation – Hardware-related metrics

The shift from high-performance GPUs to ultra-low-power MCUs encompasses the evaluation of hardware-related metrics besides accuracy, *i.e.* latency and memory, in order to assess the portability and the efficiency. As shown in Table III, the

TABLE IV
QUANTITATIVE EVALUATION ON THE TEST SET OF KITTI DATASET [6] USING THE SPLIT OF EIGEN ET AL. [7] WITH MAXIMUM DEPTH SET TO 80M.

			Lower is better ⇓				Higher is better ⇑		
Method	Resolution	Params	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
MonoDepth2 [63]	$\downarrow 32 \times 32 \uparrow$	34.5M	**0.188**	**1.724**	7.447	0.318	0.686	0.869	0.938
MonoResMatch [10]	$\downarrow 32 \times 32 \uparrow$	42.5M	0.191	2.103	**6.670**	**0.279**	**0.742**	**0.896**	**0.953**
μPyD-Net	32×32	0.1M	0.215	2.395	7.252	0.301	0.696	0.866	0.939

TABLE V
EVALUATION OF μPYD-NET AND QUANTIZED VARIANTS AT DIFFERENT RANGES. COMPARISON WITH STATE-OF-THE-ART [10] ON THE SAME RANGES.

			Lower is better ⇓				Higher is better ⇑		
Res.	Range	Precision	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
32×32	0-80m	float32	0.215	2.395	7.256	0.302	0.695	0.865	0.939
		int8	0.498	11.712	15.133	0.656	0.439	0.714	0.850
		int8-ft	0.219	2.478	7.379	0.307	0.687	0.861	0.937
	0-50m	float32	0.206	1.865	5.710	0.287	0.710	0.875	0.946
		int8	0.421	6.004	9.769	0.529	0.461	0.751	0.889
		int8-ft	0.209	1.928	5.809	0.292	0.702	0.872	0.943
	0-25m	float32	0.172	0.929	3.155	0.238	0.764	0.910	0.965
		int8	0.308	2.023	4.717	0.354	0.546	0.854	0.944
		int8-ft	0.174	0.939	3.179	0.240	0.758	0.908	0.964
	0-15m	float32	0.136	0.448	1.800	0.189	0.822	0.939	0.979
		int8	0.219	0.761	2.438	0.248	0.718	0.919	0.972
		int8-ft	0.138	0.457	1.815	0.191	0.817	0.938	0.978
48×48	0-80m	float32	0.193	2.308	6.943	0.277	0.736	0.890	0.953
		int8	0.322	5.919	10.648	0.394	0.615	0.837	0.925
		int8-ft	0.193	2.252	6.922	0.276	0.735	0.890	0.954
	0-50m	float32	0.182	1.685	5.308	0.261	0.751	0.901	0.960
		int8	0.285	3.329	7.266	0.343	0.636	0.860	0.942
		int8-ft	0.182	1.656	5.299	0.260	0.750	0.901	0.960
	0-25m	float32	0.149	0.748	2.843	0.212	0.804	0.932	0.975
		int8	0.220	1.208	3.629	0.261	0.716	0.914	0.967
		int8-ft	0.150	0.744	2.854	0.212	0.803	0.931	0.975
	0-15m	float32	0.118	0.348	1.601	0.168	0.856	0.956	0.986
		int8	0.166	0.501	1.967	0.199	0.821	0.948	0.983
		int8-ft	0.119	0.347	1.608	0.168	0.855	0.955	0.986
[10]	0-80m		0.096	0.673	4.351	0.184	0.890	0.961	0.981
[10]	0-50m		0.092	0.504	3.336	0.174	0.899	0.965	0.984
[10]	0-25m		0.078	0.242	1.799	0.141	0.925	0.977	0.990
[10]	0-15m		0.067	0.119	1.027	0.111	0.949	0.986	0.994

TABLE VI
QUANTITATIVE EVALUATION ON MAKE3D DATASET [30].

		Lower is better ⇓			
Method	Resolution	Abs Rel	Sq Rel	RMSE	RMSE log
MonoDepth [12] ResNet50	256×512	0.451	7.299	10.139	0.223
3Net [37] ResNet50	256×512	0.407	5.060	8.558	0.203
MonoDepth2 [63]	192×640	0.375	3.694	**8.218**	0.204
MonoResMatch [10]	384×1280	0.375	4.072	8.859	0.213
DepthHints [11]	320×1024	**0.350**	**3.385**	8.242	**0.200**
PyD-Net [1]	256×512	0.510	9.106	10.538	0.225
μPyD-Net	48×48	0.531	7.607	9.726	0.226
μPyD-Net	32×32	0.607	10.687	10.252	0.237

number of parameters of standard monocular networks exceeds by far the memory constraints of commercial MCUs, preventing the deployment on the edge and therefore a direct comparison with μPyD-Net. For this reason, this section focuses on the hardware characterization of μPyD-Net, demonstrating that the adopted architectural choices are mandatory to guarantee compliance with the limited resources of the hosting system.

Table VII reports the hardware-related metrics measured at run-time on the NUCLEO-F767ZI board: RAM usage and execution time (averaged on 100 inference runs). The proposed μPyD-Net reaches a throughput of 3.4 FPS, which can be

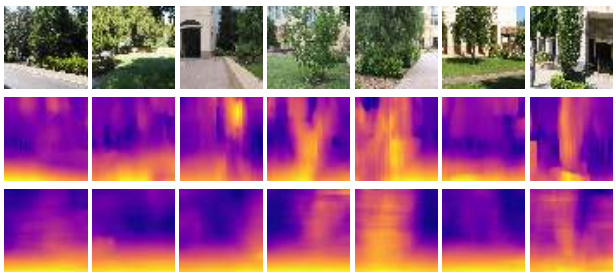

Fig. 7. Qualitative results on Make3D. From top to bottom, reference images, inverse depth maps by MonoResMatch [10] and by 48×48 μPyD-Net.

considered a remarkable result given the limited power budget of the adopted device. Moreover, the collected results demonstrate that input resolution is an effective knob in the accuracy-latency-memory space: a resolution of 32×32 enables 38% of memory savings and $2.24\times$ higher throughput compared to 48×48. This comes at the cost of some accuracy loss (as already shown in Table V). However, this might be a false problem as errors can be masked by subsequent processing stages. The resolution is a design choice indeed, and it should be weighted depending on the requirements of the downstream application.

Even though the limited computational resources of the hosting MCU prevent real-time processing even for such a compact network, the measured performance meets the requirements of the applications described in Section II. However, if higher power and area budgets are available, μPyD-Net can be ported to more powerful systems and its application extended to other use-cases. To assess the scalability of μPyD-Net from the IoT to the embedded segment, we tested its performance on the mobile CPUs (ARM Cortex-A53) adopted in our previous work [2]. In this system, μPyD-Net processes up to 320 frame/s, a $94\times$ boost that comes at the cost of $10\times$ power consumption ($\sim$4 W).

It might seem like the high efficiency brought by μPyD-Net is simply due to the input rescaling, hence our proposal may seem a relatively naive approach. A more detailed analysis reveals the design of μPyD-Net goes beyond this simplistic analysis. On the one hand, it is correct that a lower input space contributes to the reduction of the memory footprint as all the inner feature maps get intrinsically smaller. On the other hand, what makes μPyD-Net smaller and faster, hence less energy-hungry and able to fit tiny MCUs with marginal accuracy loss, lies in the topology of the network. Input resolution scaling alone is not enough, but when jointly applied on the structure of μPyD-Net it enables design options that would not be possible otherwise. Like other pyramidal architectures, μPyD-Net applies a coarse-to-fine strategy where information is processed in a hierarchical manner. Since features of higher

TABLE VII
EXTRA-FUNCTIONAL METRICS OF μPYD-NET AT DIFFERENT INPUT RESOLUTIONS ON THE NUCLEO-F767ZI BOARD.

Resolution	RAM	Execution Time
32×32	208 kB	290 ms
48×48	337 kB	651 ms

semantic level are inferred layer-by-layer traversing the pyramid bottom-up, it is intuitive to understand that the lower the resolution of the input image, the lower the number of layers needed to achieve a certain accuracy. This is a general trend also recognized in other deep learning models, but on the specific case of μPyD-Net it has a much higher impact. With smaller inputs it is in fact possible to compress the topology by reducing the number of encoders and decoders, and not just their size, thus achieving aggressive RAM reduction.

To support this analysis, the bar chart in Figure 8 shows the memory footprint vs. input resolution of PyD-Net (hatched bars) and μPyD-Net (plain bars), both quantized to 8-bit; the comparison is made splitting the contributions of weights (blue) and inner features (orange). The horizontal black line marks the RAM constraint (512 kB). For both the networks, the dimensionality of the internal activations is re-scaled according to the input size. It is worth noticing that the minimum input resolution of PyD-Net is 64×64 since the input image is down-sampled by a factor of 2^6 across the pyramidal encoders. For such reason, the results below are not reported. We can infer the following considerations. First, PyD-Net runs out of space and cannot fit into MCUs beacuse the size of the weights does not scale with the input resolution. Even using the smallest input size (*i.e.* 64×64), the weight storage is about 2 MB, namely $4\times$ the RAM on-board. Second, μPyD-Net shows an activations/weights ratio larger than PyD-Net. For instance, with the highest input resolution (256×128) the RAM taken by the features significantly increases, from 2.3 MB (PyD-Net) to 3.2 MB (μPyD-Net). The reason is that in PyD-Net the size of the feature maps processed by the topmost decoder, which is the most energy-hungry layer, is half of the input resolution, while in μPyD-Net it is not. Therefore, μPyD-Net is less suited for high resolution. However, as soon as inputs got re-scaled to 48×48, the activation footprint scales well and it meets the memory constraint. Working with 32×32 images ensures even some free space for other background applications or tasks.

These findings support our claim: μPyD-Net does work not just because of the lower cardinality of the input space, but precisely because it has been tailored to adapt to the requirements of tiny applications.

VIII. CONCLUSIONS

Depth is of paramount importance in countless practical computer vision applications and the compelling results recently obtained by frameworks aimed at inferring this cue from a single camera have dramatically increased the interest for this topic. Unfortunately, in most cases these methods require high-end GPUs or sufficiently capable embedded devices, precluding their practical deployment in several application

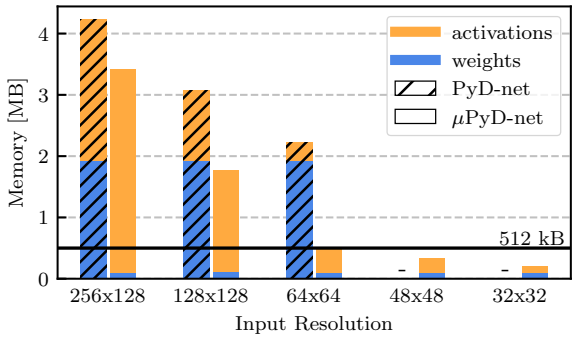

Fig. 8. Memory breakdown of PyD-Net and μPyD-Net at different input resolutions. The dash (-) indicates that the resolution is not compliant with the network topology.

contexts characterized by extreme low-power constraints such as those involving MCUs. Starting from these facts, in this paper, we proposed a two-fold strategy to enable monocular depth estimation on MCUs. At first, we designed a lightweight Convolutional Neural Network based on a pyramidal architecture, trained in a semi-supervised manner leveraging proxy-supervision obtained through a conventional stereo algorithm, capable of inferring accurate depth maps from the tiny input image fed to the network. Then, we proposed optimization strategies aimed at performing computations with quantized 8-bit data and we mapped the high-level description of the network to low-level routines suited for the target architecture. Exhaustive experimental results and an in-depth evaluation with devices belonging to the popular Arm Cortex-M family, confirm that monocular depth estimation is feasible with devices characterized by low-power constraints as MCUs. To the best of our knowledge, our method is the first one achieving this goal, fostering the deployment of monocular depth estimation to new application contexts.

REFERENCES

[1] M. Poggi *et al.*, "Towards real-time unsupervised monocular depth estimation on cpu," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 5848–5854.

[2] V. Peluso *et al.*, "Enabling energy-efficient unsupervised monocular depth estimation on armv7-based platforms," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1703–1708.

[3] R. Sanchez-Iborra *et al.*, "Tinyml-enabled frugal smart objects: Challenges and opportunities," *IEEE Circuits and Systems Magazine*, vol. 20, no. 3, pp. 4–18, 2020.

[4] D. Wofk *et al.*, "Fastdepth: Fast monocular depth estimation on embedded systems," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6101–6108.

[5] E. Chou *et al.*, "Privacy-preserving action recognition for smart hospitals using low-resolution depth images," *arXiv preprint arXiv:1811.09950*, 2018.

[6] A. Geiger *et al.*, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[7] D. Eigen *et al.*, "Depth map prediction from a single image using a multi-scale deep network," in *Advances in neural information processing systems*, 2014, pp. 2366–2374.

[8] F. Liu *et al.*, "Learning depth from single monocular images using deep convolutional neural fields," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 2024–2039, 2015.

[9] N. Yang *et al.*, "Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 817–833.

[10] F. Tosi *et al.*, "Learning monocular depth estimation infusing traditional stereo knowledge," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9799–9809.

[11] J. Watson *et al.*, "Self-supervised monocular depth hints," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 2162–2171.

[12] C. Godard *et al.*, "Unsupervised monocular depth estimation with left-right consistency," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 270–279.

[13] H. Fu *et al.*, "Deep ordinal regression network for monocular depth estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2002–2011.

[14] T. Bagautdinov *et al.*, "Probability occupancy maps for occluded depth images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2829–2837.

[15] S. Sun *et al.*, "Benchmark data and method for real-time people counting in cluttered scenes using depth sensors," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 10, pp. 3599–3612, 2019.

[16] V. Srivastav *et al.*, "Human pose estimation on privacy-preserving low-resolution depth images," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2019, pp. 583–591.

[17] M.-R. Lee *et al.*, "Vehicle counting based on a stereo vision depth maps for parking management," *Multimedia Tools and Applications*, vol. 78, no. 6, pp. 6827–6846, 2019.

[18] R. Hg *et al.*, "An rgb-d database using microsoft's kinect for windows for face detection," in *2012 Eighth International Conference on Signal Image Technology and Internet Based Systems*. IEEE, 2012, pp. 42–46.

[19] A. E. Eshratifar *et al.*, "Jointdnn: an efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing*, 2019.

[20] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[21] D. Scharstein *et al.*, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[22] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2. IEEE, 2005, pp. 807–814.

[23] L. Di Stefano *et al.*, "A fast area-based stereo matching algorithm," *Image and vision computing*, vol. 22, no. 12, pp. 983–1005, 2004.

[24] T. Kanade *et al.*, "A stereo matching algorithm with an adaptive window: Theory and experiment," *IEEE transactions on pattern analysis and machine intelligence*, vol. 16, no. 9, pp. 920–932, 1994.

[25] J. Zbontar *et al.*, "Computing the stereo matching cost with a convolutional neural network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1592–1599.

[26] N. Mayer *et al.*, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4040–4048.

[27] A. Kendall *et al.*, "End-to-end learning of geometry and context for deep stereo regression," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 66–75.

[28] J.-R. Chang *et al.*, "Pyramid stereo matching network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5410–5418.

[29] F. Zhang *et al.*, "Ga-net: Guided aggregation net for end-to-end stereo matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 185–194.

[30] A. Saxena *et al.*, "Make3d: Learning 3d scene structure from a single still image," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 5, pp. 824–840, 2008.

[31] I. Laina *et al.*, "Deeper depth prediction with fully convolutional residual networks," in *2016 Fourth international conference on 3D vision (3DV)*. IEEE, 2016, pp. 239–248.

[32] Y. Cao *et al.*, "Estimating depth from monocular images as classification using deep fully convolutional residual networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 11, pp. 3174–3182, 2017.

[33] Y. Cao *et al.*, "Monocular depth estimation with augmented ordinal depth relationships," *IEEE Transactions on Circuits and Systems for Video Technology*, 2020.

[34] H. Mohaghegh *et al.*, "Aggregation of rich depth-aware features in a modified stacked generalization model for single image depth estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 3, pp. 683–697, 2018.

[35] K. Karsch *et al.*, "Depth transfer: Depth extraction from video using non-parametric sampling," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 11, pp. 2144–2158, 2014.

[36] T. Zhou *et al.*, "Unsupervised learning of depth and ego-motion from video," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1851–1858.

[37] M. Poggi *et al.*, "Learning monocular depth estimation with unsupervised trinocular assumptions," in *2018 International Conference on 3D Vision (3DV)*. IEEE, 2018, pp. 324–333.

[38] R. Mahjourian *et al.*, "Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints," in *CVPR*, 2018.

[39] H. Kumar *et al.*, "Depth map estimation using defocus and motion cues," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 5, pp. 1365–1379, 2018.

[40] C. Wang *et al.*, "Learning depth from monocular videos using direct methods," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2022–2030.

[41] Z. Yin *et al.*, "Geonet: Unsupervised learning of dense depth, optical flow and camera pose," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1983–1992.

[42] R. Garg *et al.*, "Unsupervised cnn for single view depth estimation: Geometry to the rescue," in *European conference on computer vision*. Springer, 2016, pp. 740–756.

[43] A. Pilzer *et al.*, "Refine and distill: Exploiting cycle-inconsistency and knowledge distillation for unsupervised monocular depth estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9768–9777.

[44] L. Andraghetti *et al.*, "Enhancing self-supervised monocular depth estimation with traditional visual odometry," in *2019 International Conference on 3D Vision (3DV)*. IEEE, 2019, pp. 424–433.

[45] J. Qiu *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 26–35.

[46] H. Alemdar *et al.*, "Ternary neural networks for resource-efficient ai applications," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 2547–2554.

[47] M. Rastegari *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[48] M. Rusci *et al.*, "Quantized nns as the definitive solution for inference on low-power arm mcus? work-in-progress," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, 2018, pp. 1–2.

[49] L. Lai *et al.*, "Enabling deep learning at the iot edge," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–6.

[50] S. Vogel *et al.*, "Efficient hardware acceleration of cnns using logarithmic data representation with arbitrary log-base," in *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 1–8.

[51] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[52] M. Grimaldi *et al.*, "Optimality assessment of memory-bounded convnets deployed on resource-constrained risc cores," *IEEE Access*, vol. 7, pp. 152 599–152 611, 2019.

[53] J. Yu *et al.*, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 548–560.

[54] H. Li *et al.*, "Pruning filters for efficient convnets," in *5th International Conference on Learning Representations (ICLR)*, 2017.

[55] S. Elkerdawy *et al.*, "Lightweight monocular depth estimation model by joint end-to-end filter pruning," in *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019, pp. 4290–4294.

[56] A. Tonioni *et al.*, "Unsupervised adaptation for deep stereo," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1605–1613.

[57] A. Tonioni *et al.*, "Unsupervised domain adaptation for depth prediction from images," *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[58] A. B. Owen, "A robust hybrid of lasso and ridge regression," *Contemporary Mathematics*, vol. 443, no. 7, pp. 59–72, 2007.

[59] X. Guo *et al.*, "Learning monocular depth by distilling cross-domain stereo networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 484–500.

[60] A. Mishra *et al.*, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," in *International Conference on Learning Representations*, 2018.

[61] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[62] M. Cordts *et al.*, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[63] C. Godard *et al.*, "Digging into self-supervised monocular depth estimation," in *Proceedings of the IEEE international conference on computer vision*, 2019, pp. 3828–3838.

[64] Y. Wang *et al.*, "Unos: Unified unsupervised optical-flow and stereo-depth estimation by watching videos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8071–8081.

[65] E. Ilg *et al.*, "Occlusions, motion and depth boundaries with a generic network for disparity, optical flow or scene flow estimation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 614–630.

[66] Nucleo-f767zi. [Online]. Available: https://www.st.com/en/evaluation-tools/nucleo-f767zi.html

[67] Stm32f767zit6-datasheet. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32f767zi.pdf

[68] A. Torralba *et al.*, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008.

[69] A. Torralba *et al.*, "Object and scene recognition in tiny images," *Journal of Vision*, vol. 7, no. 9, pp. 193–193, 2007.

Valentino Peluso Valentino Peluso received the M.Sc. degree in Electronic Engineering and the Ph.D. degree in Computer Engineering both from Politecnico di Torino. He is currently a Postdoctoral researcher in the Department of Control and Computer Engineering, Politecnico di Torino. His main research interests focus on the optimization and compression of deep learning models for their deployment on low-power embedded systems.

Antonio Cipolletta holds a Master degree in Computer Engineering from Politecnico di Torino (2018). He also received the M.Sc. in Electrical and Computer Engineering from University of Illinois at Chicago (2019). He is currently pursuing a Ph.D. degree with the Department of Control and Computer Engineering at Politecnico di Torino. His main research interests focus on hardware-software co-design for emerging computing paradigms.

Andrea Calimera took the M.Sc. degree in Electronic Engineering and the Ph.D. degree in Computer Engineering both from Politecnico di Torino. He is currently an Associate Professor of Computer Engineering at Politecnico di Torino. His research interests cover the areas of design automation of digital circuits and embedded systems with emphasis on optimization techniques for low-power and reliability, energy/quality management, logic synthesis, and emerging computing paradigms.

Matteo Poggi received Master degree in Computer Science and PhD degree in Computer Science and Engineering from University of Bologna in 2014 and 2018 respectively. Currently, he is a Post-doc researcher at Department of Computer Science and Engineering, University of Bologna. His research interests include deep learning for depth estimation and embedded computer vision.

Fabio Tosi received the Master degree in Computer Science and Engineering at Alma Mater Studiorum, University of Bologna in 2017. He is currently in the PhD program in Computer Science and Engineering of University of Bologna, where he conducts research in deep learning and depth sensing related topics.

Filippo Aleotti received the Master degree in Computer Science and Engineering at Alma Mater Studiorum, University of Bologna in 2018. He is currently in the PhD program in Structural and Environmental Health Monitoring and Management (SEHM2) of University of Bologna, where he conducts research in deep learning for depth sensing.

Stefano Mattoccia received a Ph.D. degree in Computer Science Engineering from the University of Bologna in 2002. Currently he is an associate professor at the Department of Computer Science and Engineering of the University of Bologna. His research interest is mainly focused on computer vision, depth perception, embedded vision and deep learning.