Energy-Quality Scalable Monocular Depth Estimation on Low-Power CPUs

Energy-Quality Scalable Monocular Depth Estimation on Low-Power CPUs

Antonio Cipolletta[◇], Valentino Peluso[◇], Andrea Calimera[◇],
Matteo Poggi[⋆], Fabio Tosi[⋆], Filippo Aleotti[⋆], Stefano Mattoccia[⋆]
[◇]*Politecnico di Torino, 10129 Italy* [⋆]*Università di Bologna, 40136 Italy*

***Abstract*—The recent advancements in deep learning have demonstrated that inferring high-quality depth maps from a single image has become feasible and accurate thanks to Convolutional Neural Networks (CNNs), but how to process such compute- and memory-intensive models on portable and low-power devices remains a concern. Dynamic energy-quality scaling is an interesting yet less explored option in this field. It can improve efficiency through opportunistic computing policies where performances are boosted only when needed, achieving on average substantial energy savings. Implementing such a computing paradigm encompasses the availability of a scalable inference model, which is the target of this work. Specifically, we describe and characterize the design of an Energy-Quality scalable Pyramidal Network (EQPyD-Net), a lightweight CNN capable of modulating at run time the computational effort with minimal memory resources. We describe the architecture of the network and the optimization flow, covering the important aspects that enable the dynamic scaling, namely, the optimized training procedures, the compression stage via fixed-point quantization, and the code optimization for the deployment on commercial low-power CPUs adopted in the edge segment. To assess the effect of the proposed design knobs, we evaluated the prediction quality on the standard KITTI dataset and the energy and memory resources on the ARM Cortex-A53 CPU. The collected results demonstrate the flexibility of the proposed network and its energy efficiency. EQPyD-Net can be shifted across five operating points, ranging from a maximum accuracy of 82.2% with $0.4\,\mathrm{Frame/J}$ and up to 92.6% of energy savings with 6.1% of accuracy loss, still keeping a compact memory footprint of $5.2\,\mathrm{MB}$ for the weights and $38.3\,\mathrm{MB}$ (in the worst-case) for the processing.**

***Index Terms*—Monocular Depth Estimation, Energy-Quality Scaling, Embedded Systems, Low-Power CPUs, Convolutional Neural Networks, Deep Learning.**

I. Introduction

Inferring depth cues plays a crucial role in many computer vision tasks powering real-world applications like autonomous and assisted driving, self-navigation, virtual and augmented reality. Active sensors, e.g., Time-of-Flight (ToF) and Light Detection and Ranging (LiDaR), represent viable solutions, but they are expensive, significantly sized, and they also perturb the surrounding environment. A much more appealing method consists of inferring depth from images acquired with conventional passive image sensors, i.e., integrated cameras, already available in many smart devices. In this context, estimating depth from a stereo acquisition pipeline has recently achieved compelling advancements thanks to deep learning [1], [2], [3], [4]. However, there is a strong interest in adopting a more ubiquitous and cheaper monocular setup, where dense depth predictions are inferred from a still image. Methods and tools for monocular depth estimation are a promising solution for the Internet-of-Things (IoT), as they can cut the implementation cost, improve portability, and minimize energy consumption. For these reasons, monocular methods can benefit a broad spectrum of applications, such as augmented reality and virtual reality (AR/VR) on portable IoT devices adopted in smart automation systems [5], [6], [7], [8].

Pushing depth perception through the end-nodes could reduce the network congestion and increase the energy efficiency during machine-to-machine (M2M) communication processes, as it lowers the volume of data exchanged. Moreover, it also represents a practical way to guarantee users' privacy as data stay local, and it is complementary to other methods aiming to improve the reliability and the security of the whole IoT infrastructure [9], [10]. However, how to meet the tight resource constraints of the end-nodes while preserving quality-of-service still represents a considerable concern. Despite the recent improvements in monocular techniques introduced with deep Convolutional Neural Networks (CNNs), the problem of estimating depth from a single image is ill-posed and hence usually requires huge computational power to bridge the gap with the accuracy of geometry-aware methods, like stereo [11]. Prior works, e.g., [12], have relied on high-performance GPUs/CPUs indeed, which are expensive and too power-hungry for mobile/IoT applications, whereas algorithmic level optimizations offer a viable path to squeeze the requirements.

The optimization of CNNs for low-power embedded devices is a well-established problem in many application contexts [13], [14], [15], [16], [17]. The recent literature has proposed several solutions, from the designing of hardware-friendly neural architectures tailored for a specific task, depth sensing in this case [18], to model-agnostic compression strategies, like weight pruning [19], or novel resource management policies for efficient computing [20]. Obviously, a combination of them helps to reach higher efficiency. For instance, in our previous work [21], we demonstrated that the adoption of the small architecture named Pyramidal Depth Convolutional Neural Network (PyD-Net) [18], together with algorithmic and code optimizations, do enable an efficient processing of monocular depth estimation on low-cost, off-the-shelf, and low-power CPUs. Thanks to its compact topology and the adoption of reduced arithmetic precision to 8-bit, the weights of PyD-Net can fit into $1.9\,\mathrm{MB}$, achieving $\times 66.5$ memory reduction compared to other state-of-the-art monocular techniques [12]. Still, there is room for improvements as static

methods, like those aforementioned, have intrinsic limitations impeding the additional savings needed to enable monocular depth perception on tiny IoT devices.

With this work, we aim to explore the margins offered by dynamic strategies. Leveraging the adaptive error resilience of many applications, the accuracy of depth maps can be gracefully degraded for the sake of energy efficiency, depending on the specific task, the external context, or the amount of resources available at that time, still preserving high quality-of-service. A less accurate depth estimate could be sufficient when performing simple tasks, such as object and people counting, coarse-grain pose estimation, action recognition, vehicle detection, and, more in general, when the scene analyzed is "easy" and requires less effort to be understood. For instance, a robot in a free environment with few obstacles could exploit less accurate depth estimation to achieve a faster response, not least a longer battery lifetime and a longer mission time. However, a more accurate depth estimate could still be required for a more precise 3D scene reconstruction or when the scene is cluttered and more "complex". Beyond these context-driven speculations, one should consider other applications running in the background may reduce the resources available, forcing the depth estimation to be run under more stringent latency and energy constraints in certain intervals of time. This strategy, referred to as "energy-quality scaling" in the field of approximate-computing, has been already adopted for image classification tasks based on CNNs [22], but not, or very marginally, for monocular depth estimation.

As the main contribution, we introduce the design and characterization of an efficient dynamic option for energy-quality scalable depth sensing. The implementation encompasses the choice of the underlying CNN model that has to be enhanced with dynamic features. The adoption of PyD-Net as the backbone is a natural option here. As will be presented in the following sections, its modular structure can be re-scaled at run time to infer depth maps at different resolutions, hence different quality, skipping the most demanding computations when running at lower output scales. Moreover, its tiny memory footprint enables the storage of multiple instances, that is, weight-sets at different arithmetic precision, which can be dynamically selected at run time to further extend the energy-accuracy trade-off. Therefore, borrowing the solution proposed in our previous work [21], we hereby elaborate on an energy-quality scalable system named EQPyD-Net.

The dynamic energy-quality trade-off attainable with EQPyD-Net is the result of several optimizations orchestrated across the design and deployment stack, which contribute to the novelty of our proposal compared with [21] and can be summarized as follows:

- at training-time, the integration of a multi-scale self-supervised training procedure that enables to infer high-quality depth maps at five different resolutions without any architectural modification;
- at optimization-time, the use of a hardware-friendly model conversion to fixed-point arithmetic at different precision options, i.e., 16- and 8-bit;
- at compilation-time, the adoption of energy-quality proportional neural kernels for multi-precision fixed-point matrix multiplication.

All these features lead to an adaptive solution that ensures high accuracy with minimal memory footprint. A thorough assessment conducted on the ARM Cortex-A53 CPU using the KITTI dataset as benchmark demonstrates EQPyD-Net can meet the specifications of a wider spectrum of applications. Specifically, it extends the Pareto front in the energy-quality space enabling five operating points, from a maximum accuracy of 82.2% with $0.4\,\mathrm{Frame/J}$, up to 92.6% of energy savings with 6.1% of accuracy loss. With just $5.2\,\mathrm{MB}$ for the weights and $38.3\,\mathrm{MB}$ (worst-case) for the processing, EQPyD-Net represents a promising solution for the IoT.

The rest of the paper is organized as follows. Section II summarizes the main advancements in training CNNs for monocular depth estimation, the standard optimization practices to deploy depth estimation networks on embedded systems, and the existing approaches to implement energy-quality scaling in CNNs. Section III describes the PyD-Net architecture and the training pipeline integrating a multi-scale loss function. Section IV focuses on post-training optimization to translate the network into efficient code tailored for the target device, presenting the implementation of the adopted multi-precision neural kernels. Section V presents the collected results, with an extensive evaluation of the different design knobs on both functional and extra-functional metrics. Section VI outlines the scope of the proposed approach, recalling the inherent limitations of monocular depth perception. Finally, Section VII closes this paper, discussing the main findings of our study.

II. Related Works

A. Self-Supervised Monocular Depth Estimation

Early approaches for depth estimation relying on deep learning make use of supervision [23], [24], [25] achieving unpaired accuracy compared to previous works in the field [26], [27], [28]. Nonetheless, collecting large amounts of labeled images is extremely costly and often requires additional sensors and hand-made post-processing.

To overcome the need for ground truth data, CNNs can be trained by casting depth estimation as an image reconstruction problem, thus in a self-supervised manner. Two (not mutually exclusive) categories of approaches follow this direction, respectively using monocular sequences or stereo pairs. We followed the second approach, namely supervision from stereo pairs, as pioneered by [29] and [12] and improved by many other methods, simulating a trinocular setup [30], jointly learning for semantic [31], using higher resolution [32], GANs [33], sparse inputs from visual odometry [34] or uncertainty estimation [35]. Stronger supervision can be obtained through *noisy* annotations from the raw output of a LiDAR sensor [36] or traditional stereo algorithms [37], [38], [39].

The most recent work focusing on self-supervision from stereo images is MonoDepth2 [40], introducing loss computation of multi-scale predictions brought at full-resolution and allowing for higher accuracy. The novel training procedure we

Table I: Overview of the state-of-the-art CNNs for monocular depth estimation on embedded systems.

CNN	Training Loss	EQ-Scaling
FastDepth [19]	Supervised	✗
EDA [41]	Supervised	✗
PyD-Net	Self-supervised	✓

propose for EQPyD-Net in this paper is inspired by this strategy. However, Monodepth2 uses low-resolution predictions only during training and not during inference. In this work, we demonstrate that EQPyD-Net can infer low-resolution maps (up to 1/32 of the input) while keeping accuracy high.

B. Monocular Depth Estimation on Embedded Systems

Extensive research was conducted to develop deep learning techniques for depth estimation on GPUs, but only a few works dealt with the implementation and porting of such techniques onto embedded platforms. The shift from power-hungry to resource-limited computing systems poses several concerns, calling for new solutions where hardware-related metrics play as concurrent variables to optimize together with the accuracy. A comprehensive overview of state-of-the-art solutions designed for low-power applications is reported in Table I.

To meet the stringent requirements of embedded systems, FastDepth [19] and EDA [41] propose novel compact architectures tailored for mobile GPUs, like the NVIDIA Jetson TX2 board powered by a mobile version of the Pascal architecture. These methods rely on a standard supervised training procedure that requires a full annotated dataset, often not available in many real-life applications. The targets of this work are CPU-based devices with much lower power budgets. Compared to the aforementioned works, PyD-Net combines two unique advantages. First, it makes use of a self-supervised training procedure, which enables to cut the implementation costs. Second, its multi-scale architecture can be easily adapted to different resource budgets, offering the opportunity to reach a better energy-quality trade-off at run time without further modifications of the network topology. To show how to efficiently implement these features leveraging a self-supervised training procedure is one of the main contributions of this work.

C. Energy-Quality Scaling for ConvNets

Dynamic energy-quality scaling is enabled by tunable knobs deployed at the many levels of the computing stack, from circuit [42] to architecture [43] and algorithm [44], and in all system components, such as processing [42], memory [45], and I/O sub-systems [46]. Additional orthogonal knobs could arise from the specific application domain, as in the case of CNNs, where tasks show high resilience to approximation. The resilience of a CNN is the underlying principle upon which standard compression techniques are built up [47], e.g. pruning and quantization. The extension of this principle to dynamic strategies for run-time scaling has been investigated in several studies, although mainly applied to simple tasks like image classification. This section reviews the most common approaches.

Resolution scaling. It plays with the spatial resolution of the intermediate features processed by the network layers. High-resolution features may contain fine-grain details, which enrich the expressive power of the CNN, improving its prediction accuracy. However, higher resolution features need more operations to be processed; therefore, it is paramount to properly tune the resolution depending on the actual constraints. This can be achieved by scaling the resolution of the input images fed to the network [48] or producing multi-scale features that are processed by independent branches [49]. PyD-Net follows the latter strategy, in which the selective processing of the proper branches enables balancing energy and quality.

Width Modulation. These methods cut entire convolutional filters, whose number is scaled across all the layers according to a predefined ratio, called the width-multiplier [48]. More advanced solutions share the weights across the different configurations (i.e., at different values of the width-multiplier), avoiding the overhead of storing multiple weight-sets to implement a dynamic scaling [50].

Layer Skipping. It is similar to width modulation, but it operates at a coarser granularity, skipping entire convolutional layers. Pre-trained agents drive the layer selection according to contextual information extracted from the input features [51], [52]. Multi-scale networks like EQPyD-Net combine resolution scaling with layer skipping. In this case, the selection of the layers is coupled with the adopted output resolution.

Arithmetic precision. Precision scaling leverages an arithmetic relaxation of the operations to reduce the computational effort and to alleviate the memory bandwidth requirements. The scaling can be performed at different granularities, e.g., per-layer or per-network. The former assigns a different arithmetic precision to each layer [21], [53], [54], the latter operates the same scaling on the whole network. The second option is preferred for general-purpose cores, such as those targeted in this work, thanks to its simple implementation.

Besides resolution scaling, our optimization flow also includes a precision reduction strategy tailored to the multi-precision parallel arithmetic units available in the Cortex-A CPU. An in-depth analysis conducted in Section V validates our choices, showing that precision scaling further extends the achievable savings.

III. Design and Training Flow for EQPyD-Net

The proposed implementation of energy-quality scalable depth estimation encompasses the use of a pyramidal CNN and a self-supervised multi-scale training that enables the dynamic scaling mechanism. The next two subsections aim to describe these two ingredients.

A. The Underlying Pyramidal Network Architecture

Figure 1 depicts an abstract view of the underlying pyramidal CNN architecture adopted in this work. Inspired by the most popular networks for vision applications, it presents

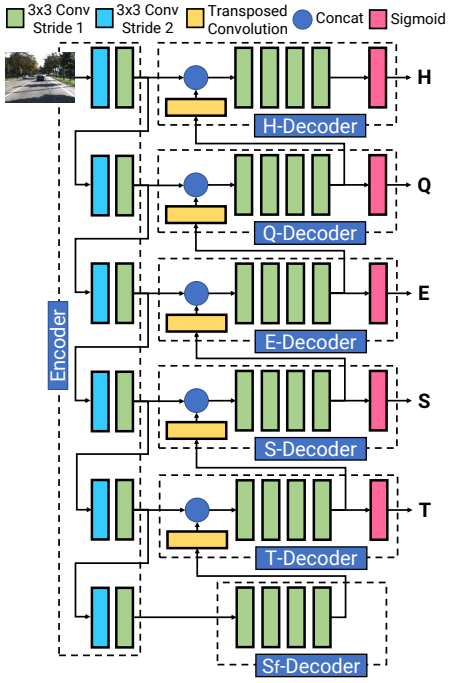


Figure 1: Schematic view of the PyD-Net architecture. Its multi-scale architecture enables to infer depth maps at different resolutions: H stands for $\frac{1}{2}$ of the input resolution, Q for $\frac{1}{4}$, E for $\frac{1}{8}$, S for $\frac{1}{16}$, T for $\frac{1}{32}$.

an encoder-decoder architecture. Such design comes from the need to extract high-level low-resolution features (encoder) and then to refine those features to generate high-resolution and dense disparity maps (decoder). The key differentiating factors from other solutions are the extremely lightweight pyramidal encoder and the modular architecture of the decoder composed of multiple branches. The pyramidal encoder, halving the resolution of the feature maps at each level, allows each decoder branch to operate at a different spatial resolution. Therefore, the decoders on the top operate on fine-grain details, while the decoders on the bottom capture the general context of the input image. Moreover, a deconvolution layer acts as a bridge between the different decoder branches by upsampling the disparity map obtained at a lower level before feeding it as input to the higher branch.

The encoder has a lightweight architecture that enables a reduction of the number of weights (hence memory) and the number of MACs operations (hence latency). Specifically, two 3×3 convolutional kernels compose each level of the encoder (from H to Sf), the former having a stride 2 for down-sampling, the latter a stride 1 for feature extraction. The number of filters increases when moving from the top to the bottom: 16, 32, 64, 96, 128, 192. For comparison with the proposed encoder, Table II reports the MACs and the number of parameters of other common encoder architectures: VGG16, ResNet-18, and MobileNet v2. PyD-Net is at least 3.4x less space-hungry and 1.93x more time-efficient.

The decoder is composed of six branches, therefore producing depth maps at six different resolutions, from $\frac{1}{2}$ (H) to $\frac{1}{64}$ (Sf) of the input resolution. Results from decoder branches at *half* H, *quarter* Q, *eighth* E, *sixteenth* S, *thirty-second* T resolution constitute the five possible outputs of the network. All decoder branches show the same number of filters and preserve the spatial resolution of the input maps. In particular, a decoder branch consists of four 3×3 convolutional layers having 96, 64, 32, and 8 filters, with a leaky ReLU as the activation function. Each decoder branch processes the concatenation of the upsampled depth map from the previous level with the pyramidal features of the same level. Since each branch consumes feature maps at a different resolution, the bottom ones require fewer operations and a smaller RAM footprint than the top ones. However, this peculiar combination of a lightweight encoder with decoder branches that require a computational effort proportional to the quality demand makes PyD-Net an excellent candidate for a scalable Energy-Quality system. Indeed, the computational burden of the high-resolution decoders is paid only in case of high output resolution. On the other hand, the encoder is always executed regardless of the final resolution; therefore, it has been designed to be extremely fast and small. Furthermore, even in the most energy-hungry configuration, PyD-Net is still more efficient than other traditional multi-resolution networks like Monodepth [12], which also shows poor scalability [18].

Table II: Number of parameters (# Params) and multiply&accumulate operations (# MACs) of the most common encoders and PyD-net encoder.

Network	**# Params (M)**	**# MACs (G)**
VGG16	138.36	40.37
ResNet-18	11.69	4.76
MobileNet v2	3.50	0.85
PyD-net Encoder	1.02	0.44

B. The Training Procedure

We follow state-of-the-art methodologies to train our network in a self-supervised manner assuming that stereo data is available during training. In particular, we train our model to minimize different losses consisting of appearance, smoothness, and a consistency term as done in [12].

Single-scale loss. For a depth map d^l estimated from an input image I^l, the loss function is obtained as a weighted sum of three main terms:

$$\mathcal{L}^{d^l} = \alpha_{ap}\mathcal{L}^{d^l}_{ap} + \alpha_{ds}\mathcal{L}^{d^l}_{ds} + \alpha_{lr}\mathcal{L}^{d^l}_{lr} \quad (1)$$

where $\mathcal{L}^{d^l}_{ap}$ is an image appearence term, $\mathcal{L}^{d^l}_{ds}$ is a smoothness term, and $\mathcal{L}^{d^l}_{lr}$ represents a left-right consistency term. Here, α_{ap}, α_{ds}, and α_{lr} represent the hyper-parameters used for controlling the importance of the each optimization term.

Appearance term. The first term measures the photometric difference between the input image I^l and the reprojected one $\tilde{I}^l$ obtained by warping the corresponding right image I^r, available during training, according to the estimated d^l:

$$\mathcal{L}^{d^l}_{ap} = \alpha \frac{1 - SSIM(I^l, \tilde{I}^l)}{2} + (1-\alpha)||I^l - \tilde{I}^l|| \quad (2)$$

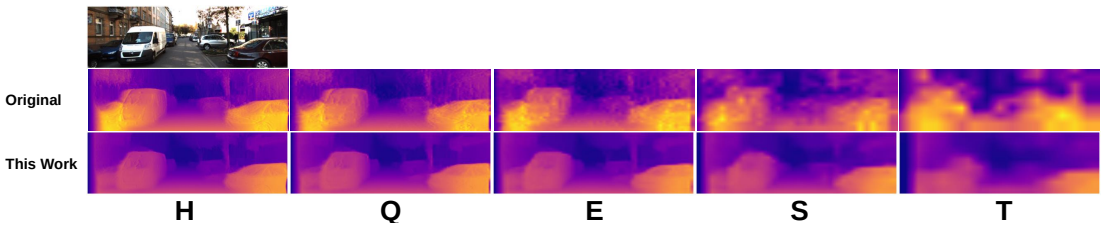


Figure 2: Depth estimated at different output resolutions for an input taken from the KITTI dataset. On top [18], on the bottom this work.

where $SSIM$ represents the Structural Similarity Index Measure [55] adopted to measure the similarity between the two images in combination with a standard ℓ_1 loss.

Smoothness term. To discourage large discontinuities in the estimated depth map, this term penalizes the difference between neighboring pixels in d^l, except where strong edges occurs in I^l:

$$\mathcal{L}_{ds}^{d^l} = |\delta_x d^l| e^{-||\delta_x I^l||} + |\delta_y d^l| e^{-||\delta_y I^l||} \tag{3}$$

where $\delta_x(.)$ and $\delta_y(.)$ are gradients in horizontal and vertical direction, respectively.

Left-right consistency term. As in [12], we train our network to estimate a second output d^r aligned with I^r and enforce consistency between d^l and reconstructed $\tilde{d}^l$ obtained by warping d^r according to d^l:

$$\mathcal{L}_{lr}^{d^l} = |d^l - \tilde{d}^l| \tag{4}$$

The three loss term can be computed over d^r as well, by replacing $d^l, I^l, \tilde{I}^l$ with $d^r, I^r, \tilde{I}^r$. Accordingly, the single-scale loss is obtained by summing the three terms computed over both d^l and d^r.

Multi-scale loss. To provide supervision to each decoder directly, a multi-scale loss function is obtained by summing up the loss signals computed at each scale s. This is traditionally achieved by downsampling I^l, I^r at each scale to compute $\mathcal{L}^{d_s}$ [12], [18]. Unfortunately, this produces sub-optimal results in terms of final accuracy as well as poor energy-quality scalability, in particular limiting to $\frac{1}{8}$ of the original resolution the lowest scale at which meaningful results can be predicted by PyD-Net, as we will see next in our experiments. For this reason, we follow the multi-scale approach revised in [40] and compute each single-scale loss signal at full resolution by upsampling each estimated depth d_s at scale s to the full resolution map $d \uparrow_s$. Then, the final loss signal $\mathcal{L}$ is obtained as the sum of each single-scale loss $\mathcal{L}^{d\uparrow_s}$

$$\mathcal{L} = \sum_{s \in [\frac{1}{2}, \ldots, \frac{1}{32}]} \mathcal{L}^{d\uparrow_s} \tag{5}$$

Fig. 2 gives a preliminary comparison between results obtained respectively by the original training adopted in [18] for PyD-Net and the new one deployed in this work for EQPyD-Net, highlighting that the former can only provide meaningful estimations down to $\frac{1}{8}$ of the input resolution while EQPyD-Net reaches down to $\frac{1}{32}$ while still allowing for scene understanding.

IV. Optimization

The efficient deployment of CNNs on low-power embedded systems requires careful model optimization to deal with the limited available resources. In this regard, quantization to fixed-point arithmetic is a de-facto standard since it brings several advantages: (i) reduces the memory footprint of the model; (ii) accelerates performance thanks to better utilization of caches and memory bandwidth. Lower precision could potentially bring larger savings at the cost of lower quality. However, quantization alone is not enough as actual savings can be achieved only through a proper implementation of fixed-point convolutions in the hosting hardware [56], [57], [58]. This implementation requires a smart orchestration of the processing units to efficiently exploit the higher throughput brought by bit-width scaling.

To achieve this goal, we developed a new set of integer kernels (*Q.Neural-Kernels*) optimized for 16- and 8-bit inference. These kernels are tailored to the Cortex-A architecture to maximize the utilization of the single-instruction multiple-data (SIMD) unit embedded in modern ARM CPUs. Thanks to dedicated code optimization, the proposed kernels guarantee that the energy needed for inference is proportional to the bit-width adopted, allowing energy-quality scaling through precision reduction.

The Q.Neural Kernels were integrated into the optimization flow illustrated in Fig. 3. The flow involves two main stages: (i) the front-end, which takes as input the PyD-Net trained with 32-floating-point (FP32 in the figure) and returns the fixed-point models (16- and 8-bit); (ii) the back-end, which translates the high-level description of network in low-level code that can efficiently run on the target hardware.

A. Model Optimization: Fixed-Point Quantization

Quantization to fixed-point arithmetic encompasses the definition of an affine mapping of integers q (represented with 16 or 8 bits) to floating-point numbers r:

$$r = K \cdot (q - q_0) \tag{6}$$

where K is the scale factor, and q_0 is the quantized value corresponding to the floating-point value 0.

In this work, we resorted to a linear quantization based on symmetric binary scaling: q_0 is fixed to 0 (symmetric scaling); the scale factor K is constrained to a power of 2, i.e., $K = 2^{-p}$, where p is the position of the radix-point in q (binary scaling). In the adopted scheme, all the layers

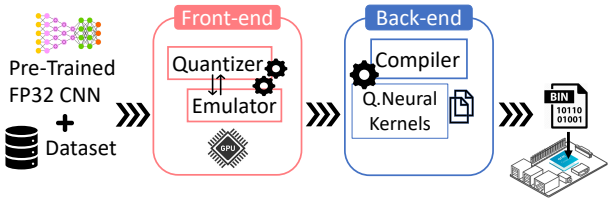


Figure 3: Optimization flow for EQPyD-Net quantization and deployment on ARM Cortex-A.

share the same bit-width (16-bit or 8-bit), while the radix-point is assigned layer by layer to match the different value distributions across the network. This design choice allows for a proper organization of the low-level code and efficient management of the hardware resources. A greedy heuristic procedure identifies for each layer the radix-point position that minimizes the mean squared error between the original floating-point values and the quantized ones.

Although other quantization strategies, e.g., asymmetric scaling [59], may achieve higher accuracy at the cost of additional computational stages, in the case of EQPyD-Net, a linear symmetric binary scaling quantization reaches the same accuracy of floating-point, making other more complex schemes irrelevant.

Working with fixed-point CNNs needs proper tools that emulate integer arithmetic units during the training and validation stages. To this purpose, we built an in-house emulation engine that leverages the acceleration of GP-GPUs. The engine integrates the fake-quantization method introduced in [59]. During inference, a software wrapper converts activations and weights (stored in fixed-point) to the 32-bit floating-point. Once processed, data are converted back in fixed-point and adjusted with auxiliary transformations (e.g., saturation, truncation, binary-shift) that replicate the behavior of the fixed-point units of the ARMv7 core (e.g., saturation of the accumulator register, set-up of the radix-point position).

A fine-tuning stage is operated to recover the loss introduced by quantization. This procedure works as follows: (i) the forward-propagation is run with fake-quantization; (ii) the gradients are back-propagated using the straight-through estimator method [60] (iii) weights are kept in a floating-point format to allow small weight updates; (iv) weights are quantized at the end of each epoch using stochastic rounding. Compared to standard floating-point training, the execution time increases by 20% (from 10 to 12 min. per epoch). The learning flow leverages knowledge distillation; the quantized model, set as the student, is re-trained to mimic the original floating-point network, the teacher. The training loop is driven by a multi-scale loss function that minimizes the mean squared error between the disparity maps inferred by the two actors (teacher d^{F} and student d^{Q}) at the different output resolutions:

$$\mathcal{L} = \sum_{s \in [\frac{1}{2}, \ldots, \frac{1}{32}]} ||d_s^{\mathrm{Q}} - d_s^{\mathrm{F}}||^2 \qquad (7)$$

B. Multi-precision Neural Kernels for Energy-proportional Inference

The common belief that fixed-point representations reduce energy consumption due to less complex arithmetic is not exactly true. Indeed, the correct execution of multiply&accumulate operations in fixed-point asks for additional instructions not needed in floating-point, such as data extension and arithmetic shifts. Moreover, proper dispatching of the operations involved is paramount to avoid under-utilization of the local memories (register file and caches) and the execution lanes.

The Cortex-A CPU targeted in this work hosts the NEON Media Processing Engine, a programmable Single-Instruction Multiple Data (SIMD) architecture that relies on multiple arithmetic units to accelerate parallel workloads, like CNN inference. With one single instruction, the same operation is simultaneously executed over multiple data to obtain a performance boost. The NEON architecture supports both parallel floating-point and integer instructions. The register file can be configured to host 8-, 16-, 32-, 64-, or 128-bit data, while the integer data-path supports 8-, 16-, 32- or 64-bit operations.

The proposed Q.Neural Kernels leverage a custom implementation of the GEneral Matrix-Multiplication (GEMM) algorithm [61], tailored to the NEON unit. In its general embodiment, the GEMM-based implementation of a 2D convolution encompasses the transformation of the multi-dimensional input tensors (activations and weights) into two-dimensional matrices. These matrices are iteratively split into regular tiles to maximize data-reuse across the memory hierarchy. Among all possible tiling choices, we resorted to an output stationary dataflow [62] (Fig. 4a), where the output matrix is divided in tiles of shape $N_{\mathrm{x}} \times N_{\mathrm{y}}$ and each output pixel keeps stationary in a dedicated register of the register file till the end of the computations. The adopted dataflow is the most efficient choice in fixed-point convolutions, as explained later in the text.

The processing of each output pixel involves a sequence of multiply&accumulate instructions. Multiple output pixels can be computed in parallel depending on the precision adopted, 2 for 16-bit and 4 for 8-bit. A detailed example is depicted in the schematic representation of Fig. 4b. It illustrates the parallel calculation of two outputs C_{00} and C_{01}. In general, $C_{ij} = \sum_k A_{ik} \cdot B_{kj}$, with $i \in [0, N_{\mathrm{x}}), j \in [0, N_{\mathrm{y}})$. The example is for 16-bit fixed-point (the same holds for 8-bit, yet with doubled parallelism). The flow is as follows:
(1) the 16-bit (8-bit) input operands, A_{ik} and B_{kj} are extended to 32-bit (16-bit) obtaining A'_{ik} and B'_{kj} (Fig. 4b refers to B_{kj} only);
(2) the input operand A'_{ik} is broadcasted to all arithmetic units to exploit data-reuse, while the other input B'_{kj} is streamed across the units; two (four) fused multiply&accumulate (MAC) operations are executed in parallel; the result is stored into a 64-bit (32-bit) register C'_{ij};
(3) at the end of the loop, the two (four) results are ready to be packed and then stored in the main memory; an output processing stage (highlighted in green) is in charge of the radix-point

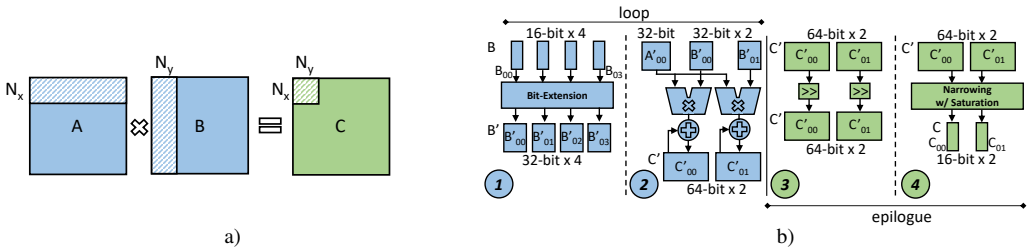


Figure 4: Q.Neural-Kernel: execution flow for 16-bit fixed-point.

shifting according to the desired radix-point position;
(4) the result is shrunk to the original bit-width, i.e., 16-bit (8-bit), and eventually saturated.

The bit-extension of step **(1)** guarantees 32- (16-) guard-bits for the accumulation. This operation is crucial as it avoids overflow/underflow during accumulation. Bypassing this stage may achieve twice the parallelism, but results are highly inaccurate. Considering the larger bit-width of the partial sums, the adoption of an output stationary flow is of paramount importance, as it reduces the number of bytes moved across the memory hierarchy.

Concerning the shape of the output tiles, we empirically found that $(N_\mathrm{x} = 6, N_\mathrm{y} = 4)$ for 16-bit and $(N_\mathrm{x} = 6, N_\mathrm{y} = 8)$ for 8-bit achieve a good balance between computing and memory intensity.

Thanks to the adopted implementation choices, the benefit of precision scaling is twofold. First, at lower precision the number of elements in the output tile doubles, halving the data movement across the memory hierarchy, therefore reducing memory energy. Second, the number of multiply&accumulate operations processed by a single instruction doubles from 2 (16-bit) to 4 (8-bit), enabling faster processing hence improved energy efficiency.

As a side note, we point out that the parallelism of floating-point is 4. Contrary to what is thought, floating-point requires fewer operations since the additional output stage (steps (3) and (4) in Fig. 4b) is not needed. Despite that, the performance of fixed-point CNNs improves over floating-point. This gain is due to the following factors: (i) enhanced utilization of memory bandwidth; (ii) smaller memory footprint for storing weights and partial results, hence less RAM usage; (iii) higher hit-rate in caches. This analysis is confirmed by experimental evidence discussed next.

V. Experimental Results

In this section, we describe the set-up for the functional evaluation and for the experiments run on-board. The collected results aim at assessing the performance of EQPyD-Net, according to several figures-of-merit, both functional (i.e., error and accuracy metrics) and extra-functional, namely, energy consumption and memory footprint. An extensive analysis is reported at the end of the section with the intent of dissecting the energy-quality scalability of EQPyD-Net enabled by the two knobs: precision and network scaling.

A. Training set-up

We extensively evaluate the energy-quality scalability of EQPyD-Net on the KITTI raw dataset [63], a widely adopted dataset for depth estimation using real images. It contains about 42K rectified stereo pairs depicting driving scenarios (i.e., cars, roads, buildings), for a total of 61 video sequences collected by a driving car. Besides stereo pairs, the dataset also provides metric depth measurements obtained using a LiDAR device mounted on top of the car. Following previous works [23], [12], we split the dataset into two subsets, with 29 and 32 sequences respectively, sampling 697 frames from the first split and 22600 from the second group for test and training purposes.

We trained the network in a self-supervised manner following the schedule described in [18] for a fair comparison with the original PyD-Net models [18], [21]. This protocol consists of 50 epochs of pre-training on the CityScapes dataset [64], followed by 200 epochs on the Eigen training split. In both cases, at each step, we processed batches of 8 images resized to 512×256. We used Adam optimizer [65] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-8}$, a learning rate of 10^{-4} for the first 60% epochs, halved every 20% epochs until the end of the training process. During training, we performed random horizontal image flipping, random color augmentation in terms of gamma correction, brightness modification, and color shifting. Each augmentation has a 50 percent chance to be applied. The values of gamma, brightness, and color shift are sampled from a uniform distribution in [0.8,1.2], [0.5,2.0], and [0.8,1.2] ranges respectively.

For what concerns the knowledge-distillation applied after quantization, we fine-tuned the network for five epochs with Adam optimizer and learning rate $1.0\mathrm{e}{-5}$ on the Eigen training split.

B. Evaluation

To evaluate the accuracy of depth estimation, we followed the standard methodology [23] in computing the error metrics (the lower the better) between the predicted and ground-truth depth, namely, the Absolute Relative Error (Abs Rel), the Squared Relative Error (Sq Rel), the Root Mean Squared Error (RMSE), and the Logarithmic Root Mean Squared Error (RMSE log). Additionally, we computed the accuracy scores (the higher the better) α_n defined as the percentage of predicted depth values whose ratio and inverse ratio with the ground truth is below a given threshold 1.25^n. For the sake of

Table III: Evaluation metrics. y denotes the predicted depth, y^* the ground-truth depth. In the evaluation, only labelled pixels are considered. Here, N represents the amount of valid pixels in the ground-truth depth map.

Notation	Definition	Equation
Abs Rel	Absolute Relative Error	$\frac{1}{N}\sum_{i=1}^{N}\frac{\lvert y_i - y_i^*\rvert}{y_i^*}$
Sq Rel	Squared Relative Error	$\frac{1}{N}\sum_{i=1}^{N}\frac{\lVert y_i - y_i^*\rVert^2}{y_i^*}$
RMSE	Root Mean Squared Error	$\sqrt{\frac{1}{N}\sum_{i=1}^{N}\lVert y_i - y_i^*\rVert^2}$
RMSE log	Logarithmic Root Mean Squared Error	$\sqrt{\frac{1}{N}\sum_{i=1}^{N}\lVert \log y_i - \log y_i^*\rVert^2}$
a_n	Prediction Accuracy	% of y_i *s.t.* $\max\left(\frac{y_i}{y_i^*}, \frac{y_i^*}{y_i}\right) < 1.25^n$, $n \in \{1,2,3\}$

clarity, we summarized the detailed definition of each metric in Table III.

C. Hardware and tools

We adopted the Raspberry Pi 3B (RPI3B) as test-bench platform to evaluate the energy efficiency of the different configurations of EQPyD-Net. The RPI3B is a commercial off-the-shelf embedded board equipped with the BCM2837 System-on-Chip, integrating a quad-core ARM Cortex-A53 CPU [66] SoC by Broadcom. The system can run at a maximum clock frequency of 1.2 GHz and embeds 1 GB of DRAM. The Q.Neural Kernels (see Section IV-B) were integrated into the ARM Compute Library (ACL) [67] version 18.05, which collects optimized floating-point kernels used as baselines in our analysis. The code was cross-compiled with *gcc-linaro* toolchain version 6.4.0-2018.05. We adopted Ubuntu Mate 16.04 (32-bit) as the operating system. We collected the performance statistics of the inference stages at the maximum operating frequency with the Google benchmark suite, version 1.5.0 [68].

D. Functional Metrics

Table IV reports a quantitative assessment of EQPyD-Net under the different error measures (Abs Rel, Sq Rel, RMSE, RMSE log) and accuracy scores (a_1, a_2, a_3), assuming a 80 m cap distance [12]. The table is organized in five sections, one for each resolution (H, Q, E, S, T); each of them includes different data types (32-bit floating-point, 16- or 8-bit fixed-point, referred to as FP32, FX16, and FX8) with optional fine-tuning (-ft) carried out after quantization. At each resolution we also report the metrics measured on the original version of PyD-Net [21] (first row of each section), which is trained with a single-scale loss.

The assessment highlights the improvement brought by the proposed multi-scale loss. Scaling down to S, EQPyD-Net outperforms the original version at H resolution in all metrics, obtaining better results with much fewer resources. Moving to T, it still achieves competitive accuracy ($a_1 = 77.8\%$) with only 4.3% loss related to H, whereas, in the original version, the accuracy falls down to 45.6%, which is an unpractical value for most applications. The benefits can be perceived also from the qualitative comparison reported in Fig. 2. Moreover, we observed that the Q configuration yields better results than H for all metrics. It is, therefore, possible to remove the H-decoder from the network at deployment time, bringing substantial energy and memory savings, as will be detailed later in the text.

Table IV: Error metrics and accuracy scores measured on the KITTI raw data using the Eigen split [23] at different scales and precision options. For each resolution, the first row refers to PyD-Net trained with the single-scale loss [21]. The best results at each resolution are highlighted in bold, while the absolute bests in red.

	Lower is better				Higher is better		
Config.	Abs Rel	Sq Rel	RMSE	RMSE log	a_1	a_2	a_3
H@FP32 [21]	0.146	1.298	5.859	0.241	80.2%	92.7%	96.8%
H@FP32	**0.135**	1.154	**5.550**	**0.234**	**82.1%**	**93.2%**	**96.9%**
H@FX16	0.136	1.157	5.556	0.235	**82.1%**	**93.2%**	**96.9%**
H@FX8	0.200	1.820	6.797	0.329	69.8%	86.9%	93.2%
H@FX8-ft	0.141	**1.152**	5.634	0.239	80.9%	92.9%	**96.9%**
Q@FP32 [21]	0.149	1.350	6.128	0.246	79.5%	92.3%	96.6%
Q@FP32	**0.135**	**1.134**	**5.505**	**0.233**	82.1%	**93.3%**	**97.0%**
Q@FX16	**0.135**	1.136	5.506	**0.233**	**82.2%**	**93.3%**	**97.0%**
Q@FX8	0.196	1.918	6.735	0.297	72.9%	88.8%	94.6%
Q@FX8-ft	0.146	1.164	5.608	0.241	80.5%	93.0%	96.9%
E@FP32 [21]	0.162	1.699	7.141	0.266	76.8%	90.7%	95.9%
E@FP32	**0.137**	**1.151**	**5.546**	**0.233**	**81.7%**	**93.1%**	**97.0%**
E@FX16	**0.137**	1.153	**5.546**	**0.233**	**81.7%**	**93.1%**	**97.0%**
E@FX8	0.264	2.964	8.252	0.324	61.1%	86.8%	94.4%
E@FX8-ft	0.145	1.157	5.675	0.241	80.1%	92.8%	96.9%
S@FP32 [21]	0.221	2.768	8.960	0.347	64.3%	84.5%	92.5%
S@FP32	**0.143**	**1.235**	**5.679**	**0.235**	**80.4%**	**92.8%**	**97.0%**
S@FX16	**0.143**	1.238	5.680	**0.235**	**80.4%**	**92.8%**	**97.0%**
S@FX8	0.215	2.176	7.062	0.291	70.6%	89.0%	95.2%
S@FX8-ft	0.155	1.335	5.843	0.240	79.6%	92.7%	**97.0%**
T@FP32 [21]	0.416	9.184	12.384	0.502	45.6%	71.8%	84.7%
T@FP32	**0.165**	**1.514**	**5.990**	**0.243**	**77.8%**	**92.2%**	**97.0%**
T@FX16	**0.165**	1.517	5.993	**0.243**	**77.8%**	**92.2%**	**97.0%**
T@FX8	0.199	1.982	6.816	0.285	70.6%	89.1%	95.5%
T@FX8-ft	0.178	1.667	6.161	0.249	76.1%	92.1%	**97.0%**

Looking at the fixed-point configurations, FX16 keeps all the metrics very close to FP32, making unnecessary further training iterations. The picture changes at FX8, where the accuracy drop gets substantial (the worst case is -20.6% for a_1 in E@FX8). However, fine-tuning the network helps to fill the gap with FX16 and FP32. This can also be perceived by observing Fig. 5 in which the 3D structure of the scene is well preserved even at the lowest scales independently from the adopted data type, with a slight deterioration at T resolution only. Moreover, it can be noticed how the FX8 data after fine-tuning achieves similar outcomes obtained using standard FP32.

Overall, these findings demonstrate the efficiency of the multi-scale loss at different resolutions. Together with precision reduction, the obtained configurations make EQPyD-Net extremely appealing for energy-quality scaling in portable devices.

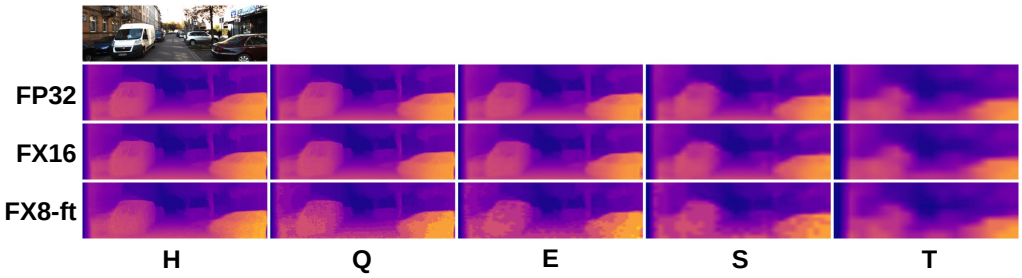


Figure 5: Depth images obtained for each value of precision and output resolution for an input taken from the KITTI dataset. The last row illustrates depth images inferred after fine-tuning (-ft).

Table V: Quantitative evaluation of KITTI test set [63] using the split of Eigen et al. [23] with maximum depth set to 80m.

		Lower is better					Higher is better	
Method	SGM	Abs Rel	Sq Rel	RMSE	RMSE log	δ <1.25	$\delta < 1.25^2$	$\delta < 1.25^3$
DepthHints [38]	✓	**0.096**	0.710	4.393	0.185	**0.890**	**0.962**	**0.981**
MonoResMatch [37]	✓	**0.096**	**0.673**	**4.351**	**0.184**	**0.890**	0.961	**0.981**
MonoDepth [12]	-	0.114	0.898	4.935	0.206	0.861	0.949	0.976
3Net [30]	-	0.111	0.849	4.822	0.202	0.865	0.952	0.978
MonoDepth2 [40]	-	0.109	0.873	4.960	0.209	0.864	0.948	0.975
PyD-Net [21]	-	0.146	1.298	5.859	0.241	0.802	0.927	0.968
EQPyD-Net (Q@FP32)	-	0.135	1.134	5.505	0.233	0.821	0.933	0.970

E. Comparison to state-of-the-art techniques

In Table V, we compare our best model Q@FP32 of Table IV to several monocular depth strategies existing in the literature on the Eigen's split of the KITTI dataset. As can be observed, the proposed training procedure applied to the original PyD-Net further improves the depth estimation accuracy and, thus, reduces the gap with respect to more complex self-supervised architectures with a larger number of parameters such as MonoDepth [12], MonoDepth2 [40], and 3Net [30]. Despite this, state-of-the-art methods leveraging other forms of supervision based on the SGM stereo algorithm during training, such as DepthHints [38] and MonoResMatch [37], still achieve better depth outcomes. However, while achieving higher accuracy, these other networks are characterized by huge computational and memory intensity, consequently requiring to be processed on high-performance CPUs or GPUs. On the other hand, EQPyD-Net, being extremely compact and offering dynamic energy-quality scaling, represents an excellent candidate for several IoT applications.

F. Energy-Quality Scaling

This section presents an extensive characterization of EQPyD-Net in the energy-quality space, aiming to understand and quantify the benefits of the adopted scaling knobs, namely resolution and precision. As quality metric we selected the a_1 score, which is commonly adopted to assess the prediction accuracy in the case of CNNs for monocular depth estimation [19]. To evaluate the energy consumed during inference, we measured the average $\mathrm{Frame/J}$ over 100 runs, assuming a constant power consumption of $3.5\,\mathrm{W}$ along the forward pass of the inference engine. Although this is a worst-case assumption, as the adopted scaling knobs could slightly lower the average power consumption, the latency variations represent the main factor in determining the energy efficiency of a CNN on a general-purpose processor.

The first analysis deals with resolution scaling. Fig. 6 shows the energy breakdown of the different modules composing EQPyD-Net. Although the pie-chart refers to the FX16 configuration, similar percentages hold for all the precision options. The key advantage of EQPyD-Net lies in the limited contribution (only 10.8%) of the modules that are always executed regardless of the selected resolution (orange area). Thanks to its lightweight pyramidal encoder, EQPyD-Net pushes the computational efforts to those decoder branches processed only when high quality is needed. Moreover, the improved training procedure allows removing the H-decoder, which is by far the most consuming block with 62.8% of energy.

The cooperation with precision scaling further extends the achievable savings, as depicted in Fig. 7. At each scale, the bar-plot illustrates the energy gains (normalized to H@FP32) achieved through fixed-point quantization. The observed trends validate the proposed Q.Neural Kernels (see Section IV-B), since the energy efficiency lowers with the precision. Most importantly, FX16 always outperforms FP32 while keeping the same level of quality. A more in-depth analysis reveals interesting aspects of the nature of the two knobs. The savings achieved by resolution scaling gets lower at smaller scales, ranging from 2.69× (H→Q at FX16) to 1.09× (S→T at FX16), whereas the savings brought by reduced bit-width keeps almost constant, around 1.38× (FX16→FX8-ft at constant resolution). Indeed, precision scaling operates as a coarse-grain knob and improves the efficiency by reducing the complexity of all the network layers.

A more intelligible representation of this concept is reported in the Pareto analysis of Fig. 8. The plot places each configuration of EQPyD-Net in the energy vs. accuracy space. The five non-dominated points are connected by the Pareto curve

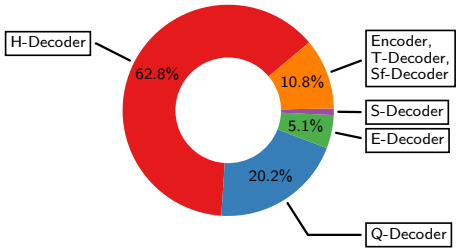


Figure 6: Energy breakdown of different modules of EQPyD-Net at FX16. Similar values have been observed also for FX8.

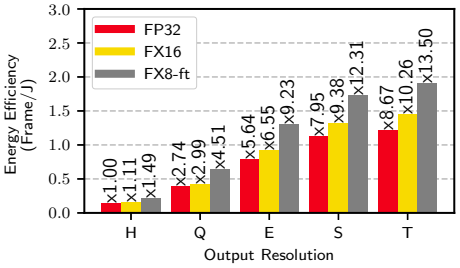


Figure 7: Energy efficiency at different output scales and precision configurations. Annotations indicate the relative improvement with respect to H@FP32 (0.141 Frame/J).

(orange line). This analysis reveals that switching across high-resolution configurations (Q, E, S) ensures significant energy savings with limited accuracy loss. Instead, when moving towards the lowest resolution (S→T), the accuracy degradation gets substantial (-2.6%) with limited energy savings (-9.38%). Conversely, precision scaling (S@FX16→S@FX8-ft) achieves 31.2% savings, yet with negligible accuracy drop (only 0.8%).

G. Memory

This subsection aims to evaluate the cost of storing the Pareto optimal configurations of EQPyD-Net (those of Fig. 8), enabling energy-quality scaling at run time. The overall memory footprint is the sum of two main contributions: the network weights, stored on the flash memory and block-loaded into RAM just before the execution, and the partial activations produced by the intermediate layers during inference. The latter are stored on the RAM through a time-shared buffer to reduce peak memory usage. At a given arithmetic precision, different resolution options share the same network weights since resolution scaling implies layer skipping (see Section III-A). By contrast, configurations centered on different arithmetic precisions need the availability of two separate weight-sets, i.e., 16-bit and 8-bit. However, as depicted in the barplot of Fig. 9a, the weights of the original PyD-Net (H@FP32) takes 7.6 MB (red bar), whereas the sum of the two configurations Q@FX16 and S@FX8-ft takes only 5.2 MB (yellow and gray bars), ensuring 44% less memory occupation.

Concerning the RAM requirements, the peak usage is dictated by the decoder branches due to their large number of channels (up to 96). Therefore, the RAM depends on the selected scale, as can be inferred from Fig. 9b. Besides that, precision scaling allows further savings with a linear reduction

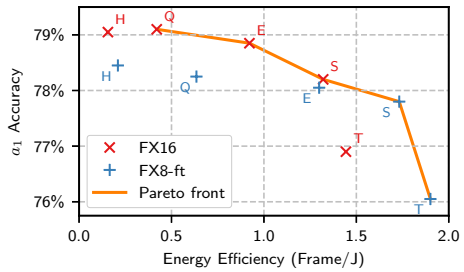
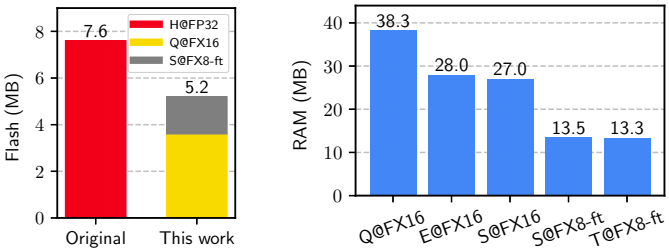


Figure 8: Energy and accuracy at different output resolutions (from left to right H→T) and precision configurations (FX16 and FX8-ft). The orange curve connects Pareto-optimal solutions.



(a) Flash (b) RAM

Figure 9: Flash (a) and RAM (b) requirements for Energy-Quality scaling with EQPyD-Net.

of RAM resources. Indeed, the transition S@FX16→S@FX8-ft halves the memory utilization. In all cases, the resources needed are limited to few tens of MBs, from 38.3 MB to 13.3 MB, instead of 206.0 MB required by the original PyD-Net (H@FP32), making EQPyD-Net extremely suitable for several embedded platforms.

VI. Current Limitations

Our approach enables flexible depth estimation, effectively allowing to trade accuracy for efficiency when deployed on platforms with low memory and energy budget. Although the single-camera setup required by our framework is cheap and thus a perfect candidate for IoT applications, single-image depth estimation remains an ill-posed problem [26], whose limitations are intrinsically inherited by our proposal. A future development aimed at overcoming these latter could aim to implement a similar, flexible solution for a binocular [11] or multi-view stereo setup.

VII. Conclusion

This work presented a thorough assessment of the energy-quality trade-off achievable by EQPyD-Net for the implementation of monocular depth estimation on low-power embedded systems. We studied the intersection of two knobs acting at different optimization levels: (i) resolution scaling through the design and training of a CNN for multi-scale inference; (ii) precision scaling through network quantization and code optimization. Experimental results collected on the Cortex-A53 CPU validate the efficiency of the adopted strategy.

Thanks to the design and optimization flow adopted, EQPyD-Net offers five working points that can be selected at run time to match different energy/quality constraints. On the KITTI dataset, EQPyD-Net reaches an 82.2% accuracy at $0.4\,\mathrm{Frame/J}$ or achieves 92.6% of energy reduction at the cost of limited accuracy degradation (-6.1%). Despite its dynamic structure, EQPyD-Net is still compact requiring only $5.2\,\mathrm{MB}$ for storing weights and $38.3\,\mathrm{MB}$ (in the worst case) for execution.

REFERENCES

[1] J. Zbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *Journal of Machine Learning Research*, vol. 17, no. 1-32, p. 2, 2016.

[2] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[3] J.-R. Chang and Y.-S. Chen, "Pyramid stereo matching network," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[4] F. Zhang, V. Prisacariu, R. Yang, and P. H. Torr, "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.

[5] S. Pirbhulal, H. Zhang, M. E. E Alahi, H. Ghayvat, S. C. Mukhopadhyay, Y.-T. Zhang, and W. Wu, "A novel secure iot-based smart home automation system using a wireless sensor network," *Sensors*, vol. 17, no. 1, p. 69, 2017.

[6] S. Pirbhulal, W. Wu, K. Muhammad, I. Mehmood, G. Li, and V. H. C. de Albuquerque, "Mobility enabled security for optimizing iot based intelligent applications," *IEEE Network*, vol. 34, no. 2, pp. 72–77, 2020.

[7] A. H. Sodhro, S. Pirbhulal, L. Zongwei, K. Muhammad, and N. Zahid, "Towards 6g architecture for energy efficient communication in iot-enabled smart automation systems," *IEEE Internet of Things Journal*, 2020.

[8] S. Dong, Z. Gao, S. Pirbhulal, G.-B. Bian, H. Zhang, W. Wu, and S. Li, "Iot-based 3d convolution for video salient object detection," *Neural computing and applications*, vol. 32, no. 3, pp. 735–746, 2020.

[9] A. Venckauskas, V. Stuikys, R. Damasevicius, and N. Jusas, "Modelling of internet of things units for estimating security-energy-performance relationships for quality of service and environment awareness," *Security and Communication Networks*, vol. 9, no. 16, pp. 3324–3339, 2016.

[10] E. Ever, P. Shah, L. Mostarda, F. Omondi, and O. Gemikonakli, "On the performance, availability and energy consumption modelling of clustered iot systems," *Computing*, vol. 101, no. 12, pp. 1935–1970, 2019.

[11] M. Poggi, F. Tosi, K. Batsos, P. Mordohai, and S. Mattoccia, "On the synergies between machine learning and binocular stereo for depth estimation from images: a survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[12] C. Godard, O. Mac Aodha, and G. J. Brostow, "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] A. H. Sodhro, A. K. Sangaiah, G. H. Sodhro, A. Sekhari, Y. Ouzrout, and S. Pirbhulal, "Energy-efficiency of tools and applications on internet," in *Computational intelligence for multimedia big data on the cloud with engineering applications*. Elsevier, 2018, pp. 297–318.

[14] Y. Lin, X. Jin, J. Chen, A. H. Sodhro, and Z. Pan, "An analytic computation-driven algorithm for decentralized multicore systems," *Future Generation Computer Systems*, vol. 96, pp. 101–110, 2019.

[15] A. McDole, M. Abdelsalam, M. Gupta, and S. Mittal, "Analyzing cnn based behavioural malware detection techniques on cloud iaas," in *International Conference on Cloud Computing*. Springer, 2020, pp. 64–79.

[16] A. McDole, M. Gupta, M. Abdelsalam, S. Mittal, and M. Alazab, "Deep learning techniques for behavioral malware analysis in cloud iaas," in *Malware Analysis using Artificial Intelligence and Deep Learning*. Springer, 2020, pp. 269–285.

[17] D. Gupta, O. Kayode, S. Bhatt, M. Gupta, and A. S. Tosun, "Learner's dilemma: Iot devices training strategies in collaborative deep learning," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. IEEE, 2020, pp. 1–6.

[18] M. Poggi, F. Aleotti, F. Tosi, and S. Mattoccia, "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.

[19] Wofk, Diana and Ma, Fangchang and Yang, Tien-Ju and Karaman, Sertac and Sze, Vivienne, "FastDepth: Fast Monocular Depth Estimation on Embedded Systems," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6101–6108.

[20] Z. Yu, P. Machado, A. Zahid, A. M. Abdulghani, K. Dashtipour, H. Heidari, M. A. Imran, and Q. H. Abbasi, "Energy and performance trade-off optimization in heterogeneous computing via reinforcement learning," *Electronics*, vol. 9, no. 11, p. 1812, 2020.

[21] V. Peluso, A. Cipolletta, A. Calimera, M. Poggi, F. Tosi, and S. Mattoccia, "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.

[22] M. Shafique, R. Hafiz, M. U. Javed, S. Abbas, L. Sekanina, Z. Vasicek, and V. Mrazek, "Adaptive and energy-efficient architectures for machine learning: Challenges, opportunities, and research roadmap," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 627–632.

[23] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Advances in neural information processing systems*, 2014, pp. 2366–2374.

[24] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *2016 Fourth international conference on 3D vision (3DV)*. IEEE, 2016, pp. 239–248.

[25] F. Liu, C. Shen, G. Lin, and I. Reid, "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.

[26] A. Saxena, M. Sun, and A. Y. Ng, "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, 2009.

[27] L. Ladicky, J. Shi, and M. Pollefeys, "Pulling things out of perspective," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 89–96.

[28] S. Fanello, C. Keskin, S. Izadi, P. Kohli, D. Kim, D. Sweeney, A. Criminisi, J. Shotton, S. B. Kang, and T. Paek, "Learning to be a depth camera for close-range human capture and interaction," *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2014*, vol. 33, July 2014.

[29] R. Garg, V. K. BG, G. Carneiro, and I. Reid, "Unsupervised cnn for single view depth estimation: Geometry to the rescue," in *European Conference on Computer Vision*. Springer, 2016, pp. 740–756.

[30] M. Poggi, F. Tosi, and S. Mattoccia, "Learning monocular depth estimation with unsupervised trinocular assumptions," in *6th International Conference on 3D Vision (3DV)*, 2018.

[31] P. Z. Ramirez, M. Poggi, F. Tosi, S. Mattoccia, and L. Di Stefano, "Geometry meets semantics for semi-supervised monocular depth estimation," in *Asian Conference on Computer Vision*. Springer, 2018, pp. 298–313.

[32] S. Pillai, R. Ambruş, and A. Gaidon, "Superdepth: Self-supervised, super-resolved monocular depth estimation," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 9250–9256.

[33] F. Aleotti, F. Tosi, M. Poggi, and S. Mattoccia, "Generative adversarial networks for unsupervised monocular depth prediction," in *15th European Conference on Computer Vision (ECCV) Workshops*, 2018.

[34] L. Andraghetti, P. Myriokefalitakis, P. L. Dovesi, B. Luque, M. Poggi, A. Pieropan, and S. Mattoccia, "Enhancing self-supervised monocular depth estimation with traditional visual odometry," in *7th International Conference on 3D Vision (3DV)*, 2019.

[35] M. Poggi, F. Aleotti, F. Tosi, and S. Mattoccia, "On the uncertainty of self-supervised monocular depth estimation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[36] Y. Kuznietsov, J. Stuckler, and B. Leibe, "Semi-supervised deep learning for monocular depth map prediction," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[37] F. Tosi, F. Aleotti, M. Poggi, and S. Mattoccia, "Learning monocular depth estimation infusing traditional stereo knowledge," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[38] J. Watson, M. Firman, G. J. Brostow, and D. Turmukhambetov, "Self-supervised monocular depth hints," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 2162–2171.

[39] A. Tonioni, M. Poggi, S. Mattoccia, and L. Di Stefano, "Unsupervised domain adaptation for depth prediction from images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 10, pp. 2396 – 2409, October 2020, available at https://doi.org/10.1109/TPAMI.2019.2940948.

[40] C. Godard, O. M. Aodha, M. Firman, and G. J. Brostow, "Digging into self-supervised monocular depth estimation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 3828–3838.

[41] X. Tu, C. Xu, S. Liu, G. Xie, and R. Li, "Real-time depth estimation with an optimized encoder-decoder architecture on embedded devices," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2019, pp. 2141–2149.

[42] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*, 2013, pp. 1–6.

[43] S. Das, D. Roberts, Seokwoo Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "A self-tuning dvs processor using delay-error detection and correction," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 4, pp. 792–804, 2006.

[44] E. Nogues, D. Menard, and M. Pelcat, "Algorithmic-level approximate computing applied to energy efficient hevc decoding," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 1, pp. 5–17, 2019.

[45] Y. Chen, X. Yang, F. Qiao, J. Han, Q. Wei, and H. Yang, "A multi-accuracy-level approximate memory architecture based on data significance analysis," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 385–390.

[46] D. J. Pagliari, E. Macii, and M. Poncino, "Approximate energy-efficient encoding for serial interfaces," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 4, May 2017. [Online]. Available: https://doi.org/10.1145/3041220

[47] A. Capotondi, M. Rusci, M. Fariselli, and L. Benini, "Cmix-nn: Mixed low-precision cnn library for memory-constrained edge devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 5, pp. 871–875, 2020.

[48] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[49] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=Hk2aImxAb

[50] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=H1gMCsAqY7

[51] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris, "Blockdrop: Dynamic inference paths in residual networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8817–8826.

[52] L. Liu and J. Deng, "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[53] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2016, pp. 1–8.

[54] V. Peluso and A. Calimera, "Energy-accuracy scalable deep convolutional neural networks: A pareto analysis," in *IFIP/IEEE International Conference on Very Large Scale Integration-System on a Chip*. Springer, 2018, pp. 107–127.

[55] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[56] V. Peluso, A. Cipolletta, F. Vaiana, and A. Calimera, "Integer convnets on embedded cpus: Tools and performance assessment on the cortex-a cores," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2019, pp. 598–601.

[57] K. Lee, J. Kong, and A. Munir, "Hw/sw co-design of cost-efficient cnn inference for cognitive iot," in *2020 Fourth International Conference On Intelligent Computing in Data Sciences (ICDS)*. IEEE, 2020, pp. 1–6.

[58] Y. Liu, Y. Wang, R. Yu, M. Li, V. Sharma, and Y. Wang, "Optimizing {CNN} model inference on cpus," in *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 2019, pp. 1025–1040.

[59] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "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.

[60] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[61] K. Goto and R. A. Geijn, "Anatomy of high-performance matrix multiplication," *ACM Transactions on Mathematical Software (TOMS)*, vol. 34, no. 3, p. 12, 2008.

[62] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[63] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[64] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[65] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[66] Bcm2837 - raspberry pi documentation. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837

[67] Arm compute library. [Online]. Available: https://github.com/ARM-software/ComputeLibrary

[68] A microbenchmark support library. [Online]. Available: https://github.com/google/benchmark

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.

Valentino Peluso Valentino Peluso received the M.Sc. degree in Electronic Engineering and the Ph.D. degree in Computer Engineering from Politecnico di Torino. He is currently a Research Assistant in Department of Control and Computer Engineering at 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.

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 reliable ICs, dynamic energy/quality management, logic synthesis, design flows and methodologies for emerging computing paradigms and technologies.

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.