Enabling Energy-Efficient Unsupervised Monocular Depth Estimation on ARMv7-Based Platforms

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

***Abstract*—This work deals with the implementation of energy-efficient monocular depth estimation using a low-cost CPU for low-power embedded systems. It first describes the PyD-Net depth estimation network, which consists of a lightweight CNN able to approach state-of-the-art accuracy with ultra-low resource usage. Then it proposes an accuracy-driven complexity reduction strategy based on a hardware-friendly fixed-point quantization. Finally, it introduces the low-level optimization enabling effective use of integer neural kernels. The objective is threefold: (i) prove the efficiency of the new quantization flow on a depth estimation network, that is, the capability to retaining the accuracy reached by floating-point arithmetic using 16- and 8-bit integers, (ii) demonstrate the portability of the quantized model into a general-purpose 32-bit RISC architecture of the ARM Cortex family, (iii) quantify the accuracy-energy tradeoff of unsupervised monocular estimation to establish its use in the embedded domain. The experiments have been run on a Raspberry PI board powered by a Broadcom BCM2837 chipset. A parametric analysis conducted over the KITTI date-set shows marginal accuracy loss with 16-bit (8-bit) integers and energy savings up to 6.55× (9.23×) w.r.t. floating-point. Compared to high-end CPU and GPU the proposed solution improves scalability.**

I. Introduction

A number of high level tasks in computer vision, such as autonomous navigation and scene understanding, leverage on 3D reconstruction. To this aim, depth estimation from 2D images plays a crucial role. Whereas active sensors such as LiDARs are expensive and provide sparse depth measurements, depth-from-images represents an attractive alternative thanks to the rapid development of new stereo and monocular techniques. These latter methods recently witnessed a dramatic boost thanks to deep learning and the possibility to train neural networks even without supervised ground-truth depth.

Monocular techniques, in particular, are an attractive solution for all those low-cost or portable applications where the use of multiple cameras would be too costly or cumbersome. The drawback is the computational power they need to achieve low latency. State-of-the-art methods rely on power-hungry GPUs indeed, a too costly option for many embedded systems where the primary constraints are power and energy consumption, form-factor, and assembling costs. Here's come the challenge: bring depth estimation onto CPU-based COTS platforms, yet preserving accuracy and performance. That's also the ultimate goal of this work, which describes design and optimization practices for deploying energy efficient unsupervised monocular depth estimation on the ARMv7-A core.

Two key factors are paramount to make the GPU-to-CPU shift success: (i) the design of a network topology able to reach high accuracy with an amount of resources compliant to that of an ordinary CPU, (ii) the availability of a vertical optimization stack for fast code optimization and effective deploying to real hardware. Concerning the first point, we borrowed the *Pyramidal Depth Network* (PyD-Net) recently introduced in [1] for unsupervised monocular depth estimation. It consists of a lightweight Convolutional Neural Network (CNN) yielding close to state-of-the-art and almost real-time performance on commercial high-end CPUs. The PyD-Net also offers multi-resolution estimation, a key feature to enable adaptive energy-accuracy scaling. Starting from this model, we propose smart strategies for network refinement, redundancy removal and software porting. The optimization stages have been integrated into an inference optimizer which delivers a low-latency/high-throughput code of the PyD-Net. The front-end runs a high-level optimization through quantization to 16-bit and 8-bit fixed-point; the method is fast, highly accurate and hardware-friendly. The back-end is in charge of the code compilation and the optimal mapping onto the target ARMv7 architecture; it leverages a new optimized version of integer neural kernels designed to exploit the computing resources of the Single Instruction Multiple Data (SIMD) datapath available in the Cortex-A architecture.

The savings achieved with the proposed design and optimization strategies bring embedded depth estimation beyond the state-of-the-art in terms of accuracy/energy-efficiency. The results obtained with the ARMv7-A core show the quantized versions of the PyD-Net (16- and 8-bit) have marginal accuracy losses and substantial speed-up w.r.t. the floating-point (32-bit). That leads to high energy efficiency, demonstrating both portability and scalability.

II. Related work

Deep learning for monocular depth estimation. While early approaches consist of predictive models based on hand-crafted features (e.g., [2], [3]) astonishing performance has been achieved through supervised deep learning techniques (e.g., [4], [5]). As an alternative to previous methods, unsupervised approaches allow replacing loss based on ground-truth depth data with much easier to obtain stereo images or monocular video sequences. Garg et al. [6] used binocular stereo pairs to train a network to predict the inverse depth that maps one image onto the other. Godard et al. [7] improved the depth estimation using the (sub-)differentiable bilinear sampler mechanism proposed in [8] and by introducing a left-right constraint, better architecture design and a more robust appearance matching loss function [9]. Poggi et al. [1] proposed PyD-Net, a novel light-weight CNN suited for real-time applications on standard high-end CPU with accuracy comparable to [7]. Concurrently, [10] and [11] improved the results achieved by stereo supervision using, respectively, a novel trinocular paradigm and adversarial loss during training. Other strategies based on stereo pairs rely on semi-supervised training data [12], [13] or enforcement of temporal consistency [14]. Finally, a more unconstrained strategy to infer depth from a single image consists in using unlabeled monocular videos for training [15], [16].

Quantization of Deep Neural Networks. Precision scaling via fixed-point quantization is an effective strategy to improve the efficiency of the inference stage. If properly implemented, the shift from 32-bit floating-point to 16- and 8-bit fixed-point [17], or even below [18], enables substantial savings with no, or marginal, accuracy loss. Previous works have been mainly validated on CNNs for image classification. Large and over-parametrized nets, such as AlexNet, ResNet, VGG, are common benchmarks, while smaller nets designed for mobile applications, e.g. SqueezeNet or MobileNet, are rarely benchmarked due to their high level of sensitivity to precision scaling [19]. Most of the works quantify the benefits brought by fixed-point at a very high-level, without assessing the impact of a real hardware implementation. The impact of quantization on complex computer vision tasks, such as object detection [20], is a less studied aspect, while depth estimation is an unexplored field. Moreover, since the PyD-Net already shows a tiny topology, its quantization is a real challenge.

From a technical viewpoint, quantization refers to the search of the fixed-point format that minimizes the accuracy loss. This encompasses the definition of the bit-width and the radix-point position. Existing techniques, mainly from the DSP theory, differ in the radix-point scaling scheme. Since a complete review is out of the scope of this work, we refer interested readers to [21]. A more important aspect for deep neural networks is the granularity of the quantization. Static approaches [22] apply the same format over the entire net; dynamic approaches, which exploit the irregular distribution of the weights along the layers, makes use of a finer selection, e.g. per-layer [23] or per-channel [24]. Hybrid strategies may use a static bit-width for the whole net and a dynamic fixed-point scaling [25]. It is worth emphasizing that a one-size-fits-all solution does not exist; efficiency is affected by the kind of neural networks and the characteristics of the underlying hardware. Worse still, there are not clear indicators to predict which strategy may reach better results and trial-and-error becomes the only practical option.

Fixed-Point Neural Networks with ARMv7. An efficient processing of fixed-point deep neural networks requires a fine organization of the assembly code. This encompasses the availability of optimized library that can support the compiling stage. ARM recently released *Computing Library* [26] an open-source repository of low-level routines that support all the basic building blocks of neural models (e.g. Activation, Convolution, Normalization, Pooling). The convolution kernels are available in floating-point, while the fixed-point version is poorly optimized. Our preliminary analysis of this library revealed fixed-point convolutions with 8-bit get slower than 32-bit floating-point. The same is confirmed in this technical report [27], where authors claim 25% performance overhead.

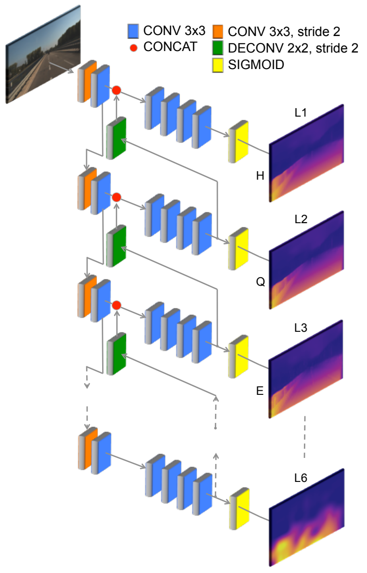

Fig. 1. PyD-Net architecture [1].

III. PyD-Net Design

PyD-Net [1] is designed to be extremely fast, efficient and suitable for real-time performance on standard CPUs. It adopts a combination of two compact functional components during spatial-pyramidal feature extraction and depth estimation. In particular, a lightweight encoder structure is used to transform the input image to a L-level pyramid of high-dimensional features, where level $L1$ represents the highest resolution (i.e., half resolution). Specifically, as depicted in Fig. 1, the pyramidal features are generated by multiple modules aimed at reducing the spatial resolution, each of them comprised of two 3×3 convolutional blocks, with stride 2 and 1. Each module produces an increasing number of channels at each scale, respectively 16, 32, 64, 96, 128 and 192, for a total of 6 levels ($L1$ - $L6$) from $\frac{1}{2}$ to $\frac{1}{64}$ of the original input resolution. Depth decoders are deployed, instead, to infer depth maps at each level of the pyramid. Specifically, starting from level $L6$, highly informative features from the pyramidal extractor are processed by four banks of filters producing, respectively, 96, 64, 32 and 8 features maps. Here a Sigmoid operator extracts a depth map for that resolution. The next level of the pyramid concatenates the features produced by the feature extractor and those up-sampled from the lower-level. The process is then repeated up to the highest resolution. Results from decoders at *half* H, *quarter* Q and *eight* E resolution enable to trade accuracy for complexity effectively. For instance, working at low-resolution E, the topmost depth decoders for Q and H are disabled thus reducing the amount of convolutions.

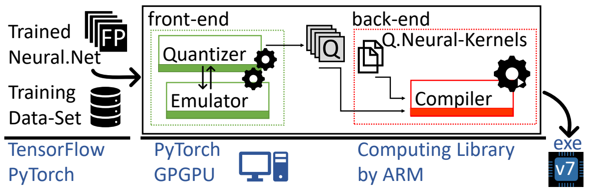

Fig. 2. Integrated flow for PyD-Net optimization and deployment on ARMv7.

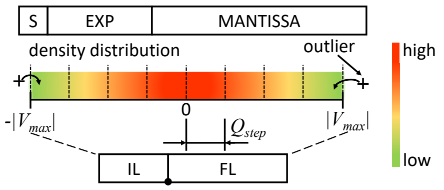

Fig. 3. Floating-Point to Fixed-point quantization.

Following [7], the network is trained on stereo pairs processing a frame as input and reprojecting the other according to estimated depth (actually, disparity). Appearance, smoothness and left-right consistency loss terms are designed for this purpose [7]. When compared with the more complex state-of-the-art encoder-decoder architecture of [7], PyD-Net [1] is about 16× smaller in size and 5× faster (at half resolution H), yet achieving comparable depth accuracy.

IV. Optimization Framework

The framework depicted in Fig. 2 is designed for a fast deploying of fixed-point PyD-Nets with ARMv7 cores. Written in Python (ver. 3.6.5), it consists of two main parts. The *front-end* takes the PyD-Net trained with 32-bit floating-point (FP in Fig. 2) and returns the fixed-point model Q. The quantization scheme is accuracy-driven and implements a hybrid strategy: the bit-width (16-bit or 8-bit, user defined) is static for all the layers, while the radix-point is assigned on a per-layer basis. This design choice allows a proper organization of the low-level code and an efficient management of the hardware resources. Incremental re-training (i.e. fine-tuning) is operated in order to recover the loss introduced by quantization.
The quantizer leverages a fixed-point emulator for fast, yet accurate loss assessment during optimization. It makes use of PyTorch libraries (ver. 0.4.1) and implements a customized version of fake-quantization [19] that works as follows. During the inference stage, a software wrapper converts data (stored in fixed-point) to floating-point. This strategy enables operations with GPU cores. 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 comparison against the results produced with the ARMv7 core revealed the emulator is highly accurate: the maximum absolute error is 8e-3 at the output of the network, with no impact on the final accuracy of the depth estimation. If compared to a standard training run, the execution time increases by 20% (from 10 to 12 min. per epoch).
The *back-end* flow, powered by the GNU Arm Embedded Toolchain [28] is fed with the fixed-point model Q and returns a binary file. A set of new integer kernels optimized for 16- and 8-bit (*Q.Neural-Kernel*) enrich the ARM Computing Library repository. Such kernels are designed to implement the proposed per-layer fixed-point scheme and to improve the utilization of the SIMD media accelerator of the ARMv7.

A. Fixed-Point Scaling

A real value V can be represented with a binary string Q of length BW (bit-width) using the following mapping function:

$$V = Q \cdot 2^{-FL} \tag{1}$$

where FL the fraction length, i.e. the position of the radix-point in Q. Given a set of real values, e.g. the weights and activations of the PyD-Net model, the choice of BW and FL affects the information loss due to quantization. This scenario is pictorially illustrated in Fig. 3, where the gradient bar reflects the density distribution of the floating-point values, $|V_{max}|$ is the largest modulus in the set, Q_{step} represents the quantization step.
Since the bit-width BW is defined by the available hardware (16 or 8 for the ARMv7), the problem reduces to searching the optimal FL (the integer length IL is then given by BW-IL). FL is constrained by $|V_{max}|$ as described in the following equation:

$$FL = \left\lfloor \log_2 \left(\frac{2^{BW-1} - 1}{|V_{max}|} \right) \right\rfloor \tag{2}$$

However, a trade-off with the quantization step Q_{step} does also exist: the smaller the FL, the larger the Q_{step}. The decision of which constraint to guard more ($|V_{max}|$ or Q_{step}) mainly depends on the distribution of the original values and their importance in the neural model.
It is worth mentioning that our problem formulation considers a symmetric distribution $[-|V_{max}|, +|V_{max}|]$; outliers are clamped to smaller values to reduce $|V_{max}|$ and improve accuracy. The adopted quantization scheme is linear (i.e. uniform intervals) with binary radix-point scaling.
Even though other quantization strategies may achieve higher accuracy, their hardware implementation results less efficient. For instance, an asymmetric method, e.g. [19], requires additional output pipeline stages that affect performance as demonstrated in [27], while floating-point scaling makes use of data-type conversions that reduce to simple shift operation with binary scaling. For the specific case of PyD-Net, our quantization strategy achieves almost the same accuracy of floating-point, making other complex schemes irrelevant.
We adopted a dynamic fixed-point scheme where the fraction length is defined layer-by-layer. More specifically, the optimization procedure aims at finding the FL_{opt} that minimizes the L2 distance between the original 32-bit floating point values X and the quantized values Q. Applied to both activations and weights independently, the procedure encompasses the following stages. First, a range analysis of weights and activations distribution; for the latter case, a subset of the training set is used (referred as *calibration set*).

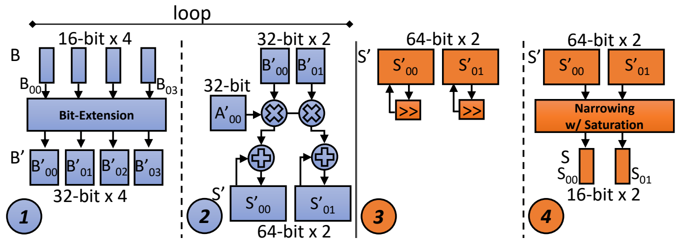

Fig. 4. Q.Neural-Kernel: execution flow for 16-bit fixed-point.

Second, extraction of lower-bound and upper-bound of the fraction length: FL_{lb}, FL_{ub}. Both are calculated with Eq. IV-A using the different V_{max} as follows:

$$V_{max} = \begin{cases} \max(|X_{min}|), |X_{max}|) & \text{for } FL_{lb} \\ \max(|X_{min}|), |X_{max}|)/K & \text{for } FL_{ub} \end{cases} \tag{3}$$

with K an arbitrary large integer[1]. Third, repeated test using the available $FL \in [FL_{min}, FL_{max}]$ and selection of FL_{opt}, i.e. the one that minimizes the L2 error. Half-even rounding is used for the quantization of trainable parameters.

B. Fixed-Point Convolution Kernels for the ARMv7

The belief that fixed-point representations reduce energy consumption due to less complex arithmetic is not exactly true. The actual benefit lies in the ability to store the same amount of information with fewer bits which in turn enables a more efficient use of the memory bandwidth [21]. However, quantization alone isn't much use. It requires a smart orchestration of the hardware processing units in order to keep pace with the higher throughput brought with bit-width scaling.

The NEON Media Processing Engine of the ARMv7-A architecture has an advanced arithmetic SIMD unit with parallel floating-point and integer units. The register file can be configured to host 8-, 16-, 32-, 64-, 128-bit data, while the integer data-path supports 8-, 16-, 32- or 64-bit operations. To explain the implemented kernel routine we resort to the example of Fig. 4, which gives a sketch of the convolution S of two matrices A and B, with B as the $N \times N$ kernel of a generic layer and A the input feature. More precisely, it illustrates the parallel calculation of two outputs S_{00} and S_{01}. In general, $S_{ij} = \sum_x^{N \times N} A_{ix} \cdot B_{xj}$. 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, $A_{i,x}$ and $B_{x,j}$ are extended to 32-bit (16-bit) obtaining $A'_{i,x}$ and $B'_{x,j}$ (Fig. 4 refers to $B_{x,j}$ only).

(2) two (four) fused multiply& accumulate (MAC) operations are executed in parallel; the result is stored into a 64-bit (32-bit) register $S'_{i,j}$

(3) after N×N loops, the two (four) results are ready to be packed and then stored in the main memory; an output processing stage (highlighted in orange) is in charge of the radix-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.

[1]We empirically verified K=100 is an optimal choice for PyD-Net

The bit-extension of step **(1)** guarantees 32- (16-) guard-bits for the accumulation. This operation is paramount as it avoids overflow/underflow during accumulation. Bypassing this stage may achieve twice the parallelism, but results are highly inaccurate. Overall, the number of parallel MAC is 2 for 16-bit and 4 for 8-bit. Further improvements are limited by the maximum bit-width of the register file: 128-bit. To notice that the parallelism of FP32 is 4. Contrary to what is thought, floating-point is intrinsically more efficient as it makes better use of the local registers. Also, it does not require the additional output stage steps (**(3)** and **(4)** in Fig. 4). Despite that, the performance of fixed-point convolutional networks improve over FP32. This is due to the following factors: (i) enhanced utilization of memory bandwidth, as the cost of accessing 16-bit (8-bit) data is half (quarter) the cost of floating-point; (ii) smaller memory footprint for storing weights and partial results, hence less RAM usage, (iii) higher hit-rate in cache. This analysis is confirmed by experimental evidence.

V. Results

A. Experimental set-up

PyD-Net and dataset. PyD-Net infers disparity maps at different resolutions thus enabling accuracy-effort scaling. The three options available, i.e. H, Q and E, have been explored for a parametric analysis of functional properties, i.e. depth accuracy, and non-functional properties, i.e. binary and RAM space, throughput and energy efficiency, under different arithmetic precision, i.e. 32-bit floating-point (**FP32**), 16-bit fixed-point (**FX16**) and 8-bit fixed-point (**FX8**). The baseline is resolution H at FP32 (H@FP32).

KITTI raw [29] is the reference dataset in this field [1], [7]. It collects 23297 images split, according to the standard protocol proposed by Eigen et al. in [30], into a training-set (22600 stereo pairs) and a test-set (697 images) with sparse ground-truth labels. The disparity maps obtained through the inference stage are transformed into depth maps following the methodology introduced by [7]. The baseline and the starting point of our work is the pre-trained PyD-Net model. It was trained, as described in [1], for 200 epochs on batches of 8 images randomly picked from the training-set and resized to 512×256.

Front-end. The quantization stage encompasses a range analysis of the activations in order to define the lower-/upper-bound of the fraction-length. For this stage, we used a calibration set filled with 5000 images randomly picked from the training-set. To notice that the intersection between testing-set and the calibration-set is void.

The fine-tuning stage (applied at post-quantization) consists of 25 training epochs made run over the full training-set using the Adam optimizer. The hyper-parameters are as follows: learning rate 1.0e-7, $\beta_1 = 0.9$ and $\beta_2 = 0.999$, weight decay = 0. The loss function and its parameters are those described in [1]. The flow is run on the NVIDIA Titan XP GPU with CUDA 8.0.

Back-end and hardware. The proposed GEMM-based Q.Neural-Kernels are written in C++ and inline assembly code.

			Lower is better		Higher is better		
Config.	Abs Rel	Sq Rel	RMSE	RMSE log	$a1$	$a2$	$a3$
H@FP32	0.146	1.298	5.859	0.241	0.802	0.927	0.968
H@FX16	0.147	1.331	5.925	0.243	0.801	0.926	0.967
H@FX16-ft	0.147	1.302	5.945	0.244	0.798	0.925	0.967
H@FX8	0.177	1.893	6.621	0.272	0.768	0.911	0.958
H@FX8-ft	0.148	1.337	6.018	0.246	0.795	0.924	0.967
Q@FP32	0.149	1.350	6.128	0.246	0.795	0.923	0.966
Q@FX16	0.149	1.365	6.155	0.246	0.794	0.923	0.966
Q@FX16-ft	0.149	1.342	6.176	0.248	0.790	0.921	0.966
Q@FX8	0.183	2.364	7.457	0.270	0.766	0.908	0.957
Q@FX8-ft	0.158	1.427	6.290	0.257	0.778	0.917	0.964
E@FP32	0.162	1.699	7.141	0.266	0.768	0.907	0.959
E@FX16	0.165	1.770	7.327	0.271	0.762	0.904	0.957
E@FX16-ft	0.162	1.712	7.163	0.266	0.768	0.907	0.958
E@FX8	0.193	2.758	8.507	0.288	0.745	0.893	0.950
E@FX8-ft	0.171	1.829	7.430	0.276	0.751	0.901	0.956

TABLE I
EXPERIMENTAL RESULTS CONCERNING DEPTH ESTIMATION ACCURACY. COMPARISON BETWEEN ORIGINAL PYDNET [1] (FP32) AND OPTIMIZED ARCHITECTURES AT DIFFERENT RESOLUTIONS.

They are integrated into the ARM Compute Library version 18.05 built with `scons` ver. 2.4.1 and the *gcc-linaro* toolchain ver. 6.4.0-2018.05. To notice that other existing libraries do not support the dynamic fixed-point scheme adopted in this work. The embedded platform used as test-bench is the Raspberry PI 3B loaded with a 32-bit Ubuntu Mate 16.04. The board hosts a quad-core BCM2837 chip-set powered to 1.2V. The four CPU cores, which belong to the ARMv8-A family, support the ARMv7-A 32-bit instruction-set in backward compatibility mode. The reason behind the adoption of the ARMv7 architecture lies in the possibility of using the depth estimation on a wider range of systems, e.g. the older Raspberry PI 2, which show lower power consumption. All the power-management features were disabled during the experimental campaign; the board drains $3.5\,\mathrm{W}$ under full utilization (4 cores ON and maximum workload); moving from FP32 to FX16 or FX8 has no effect on the total power consumption, dominated by memory accesses.

B. Accuracy vs Quantization

Table I reports an evaluation of several variants of the original PyD-Net architecture. We point out once more that the model by Godard et al. [7], counting $15\times$ parameters with respect to PyD-Net (30 vs 1.9 million), is not suited for embedded devices and not used as benchmark for non-functional properties in the next section[2]. We discuss in detail the effects introduced at each resolution (H, Q, E) by different data types (floating point, 16 or 8 bit fixed point, referred to FP32, FX16 and FX8) and optional fine-tuning (-ft) carried out after quantization, reporting error and accuracy metrics commonly adopted for evaluating depth-from-mono performance [30], assuming a 80m cap distance [7].

Looking at the table, one can notice a similar trend for the three resolution H, Q, and E, for all the seven metrics. The 16-bit fixed-point quantization FX16 introduces a negligible accuracy drop if compared to the original PyD-Net FP32. For each of the three resolutions, the additional fine-tuning (-ft) has a marginal impact on the performance; that's due to the

[2]For the sake of space, please refer to [1] for a comparison with Godard et al. [7]. For the same reason, parameters a1, a2 and a3 correspond, respectively, to $\delta < 1.25$, $\delta < 1.25^2$ and $\delta < 1.25^3$ in [1].

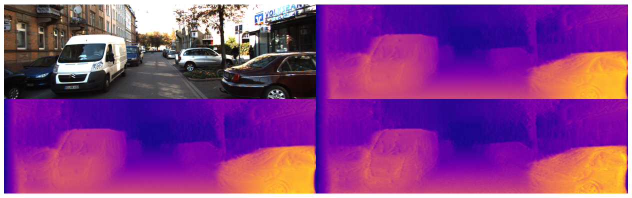

Fig. 5. Top row: Input image from KITTI dataset (left) and depth map H@FP32 computed by the original PyD-Net network [1] (right). Bottom row: depth maps H@FX16-ft (left) and H@FX8-ft (right).

fact that FX-16 is already very close to FP32. The 8-bit fixed-point quantization FX8 is more prone to accuracy drop, with substantial loss for Q and E. However, fine-tuning the network dramatically improves performance, thus closing the gap with FX16 quantization and, most notably, with the original FP32 strategy. These facts can also be perceived by Fig. 5.

C. Performance, Memory Space and Energy Efficiency

Table II collects the hardware-related metrics measured during the test-run: memory space (for weights storage), RAM usage (during PyD-Net processing) and energy efficiency (average frames per J). The throughput (frames/second) can be derived by multiplying energy efficiency (frames/J) by the total power ($3.5\,\mathrm{W}$).

As expected, energy efficiency improves when working at lower resolutions: the upper depth estimators of the net are disabled alleviating the workload (please refer to Section III). For instance, using FP32 the improvement from high (H@FP32) to low resolution is $5.64\times$. That's the savings brought by the reconfigurable topology of the PyD-Net. Considering the same resolution, energy efficiency gets larger with smaller data representations. For instance, at high resolution the gain from H@FP32 to H@FX8 is 49%; at low resolution, the gain grows up to 63.7%. To notice that, as previously outlined, moving from floating-point to fixed-point affects accuracy only marginally. The combined action of resolution scaling and precision scaling enables even larger optimization: from H@FP32 ($0.141\,\mathrm{Frames/J}$) to E@FX8 ($1.299\,\mathrm{Frames/J}$) the energy efficiency increases by $9.23\times$.

This first analysis gives clear evidence of the scaling properties of the quantized PyD-Net model, the benefits of multiple precision arithmetic, and the effectiveness of the porting flow on the target architecture. A sensing technology with such ability to implement accuracy-energy scaling represents a practical option for adaptive embedded systems: contexts or applications which tolerate lower accuracy might pursue higher energy efficiency by tuning resolution (coarse-knob) and precision (fine-knob).

Concerning memory, both the binary space and the RAM usage are important metrics that reflect the efficiency of the proposed implementation. As expected, the space for storing network parameters reduces linearly with the precision, e.g. from $7.6\,\mathrm{MB}$ with H@FP32 to $1.9\,\mathrm{MB}$ with H@FX8. At low resolution and low precision the memory space is just $1.7\,\mathrm{MB}$ (E@FX8), which brings the overall savings w.r.t.

Resolution	Precision	Space (MB)	RAM (MB)	Frame/J
H	FP32	7.6	206	0.141 (×**1.00**)
	FX16	3.8	118	0.156 (×**1.11**)
	FX8	1.9	62	0.210 (×**1.49**)
Q	FP32	7.2	60	0.386 (×**2.74**)
	FX16	3.6	34	0.420 (×**2.99**)
	FX8	1.8	19	0.635 (×**4.51**)
E	FP32	6.8	53	0.794 (×**5.64**)
	FX16	3.4	28	0.922 (×**6.55**)
	FX8	1.7	14	1.299 (×**9.23**)

TABLE II
NON-FUNCTIONAL METRICS OF PYD-NET AT DIFFERENT RESOLUTIONS AND PRECISIONS ON ARMV7-A

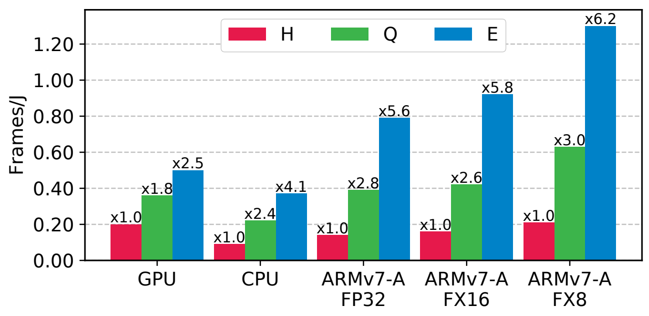

Fig. 6. Energy efficiency vs. Resolution using GPU, CPU and the ARMv7-A.

H@FP32 up to ×4.5. Even more interesting is the analysis of the RAM. Its utilization is dramatically reduced with an overall scaling factor of 14.7×: from H@FP32 (206 MB) to E@FX8 (14 MB). Indeed, the same PyD-Net compiled using prior kernels available in the ACL library showed fixed-point implementations are more resource hungry than floating-point. As a final remark, Fig. 6 provides a technology comparison among different hardware options: a GPU (Titan X Maxwell), a high-end CPU (Intel i7-6700K CPU), and the ARMv7 at FP32, FX8 and FX16. The bar chart shows the energy efficiency (Frames/J) for all the possible permutation of resolution, precision and hardware; the labels refer to the normalization w.r.t. high resolution (H) for each hardware option separately. Moving from H to E with the ARMv7@FX8 improves energy by 6.2×. To notice that ARMv7@FX16 and ARMv7@FX8 outperform both the GPU and CPU by far. A more interesting aspect the lower the arithmetic precision, the larger the gain brought by resolution scaling. While in GPU and CPU the gain from H to E is limited to ×2.5 and ×4 respectively, it grows to 5.6× with the ARMv7@FP32 and 6.2× with the ARMv7@FX8. This feature might open interesting optimization problems for resource management at run-time.

VI. CONCLUSIONS

This work introduces a comprehensive design&optimization framework aimed at improving the energy efficiency of depth perception on low power embedded devices. The target is the ARMv7, a RISC architecture widely adopted for low-cost systems. Compared to a high-end CPU (Intel i7) and a GPU (NVIDIA Titan X), the proposed implementation reaches higher energy efficiency with a negligible accuracy degradation that is tolerable by many embedded applications. More interestingly, the joint co-operation between (i) the design of the tiny, yet reconfigurable PyD-Net and (ii) the optimization enabled by the hardware-friendly fixed-point quantization allows achieving a scalability that goes beyond the state-of-the-art. These features pave the way to a widespread deployment of adaptive energy-accuracy monocular 3D sensing in additional application domains constrained by stringent energy requirements.

REFERENCES

[1] M. Poggi *et al.*, "Towards real-time unsupervised monocular depth estimation on cpu," in *IROS*, 2018.
[2] L. Ladicky *et al.*, "Pulling things out of perspective," in *CVPR*, 2014.
[3] K. Karsch *et al.*, "Depth transfer: Depth extraction from video using non-parametric sampling." *PAMI*, vol. 36, no. 11, pp. 2144–2158, 2014.
[4] H. Fu *et al.*, "Deep ordinal regression network for monocular depth estimation," in *CVPR*, 2018.
[5] Y. Luo *et al.*, "Single view stereo matching," in *CVPR*, 2018.
[6] R. Garg *et al.*, "Unsupervised cnn for single view depth estimation: Geometry to the rescue," in *ECCV*, 2016.
[7] C. Godard *et al.*, "Unsupervised monocular depth estimation with left-right consistency," in *CVPR*, 2017.
[8] M. Jaderberg *et al.*, "Spatial transformer networks," in *NIPS*, 2015.
[9] Z. Wang *et al.*, "Image quality assessment: From error visibility to structural similarity," *Trans. Img. Proc.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.
[10] M. Poggi *et al.*, "Learning monocular depth estimation with unsupervised trinocular assumptions," in *3DV*, 2018.
[11] F. Aleotti *et al.*, "Generative adversarial networks for unsupervised monocular depth prediction," in *3DRW*, 2018.
[12] Y. Kuznietsov *et al.*, "Semi-supervised deep learning for monocular depth map prediction," in *CVPR*, 2017.
[13] N. Young *et al.*, "Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry," in *ECCV*, 2018.
[14] H. Zhan *et al.*, "Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction," in *CVPR*, 2018.
[15] Z. Yin *et al.*, "Geonet: Unsupervised learning of dense depth, optical flow and camera pose," in *CVPR*, 2018.
[16] R. Mahjourian *et al.*, "Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints," in *CVPR*, 2018.
[17] J. Qiu *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *FPGA*. ACM, 2016, pp. 26–35.
[18] B. Moons *et al.*, "Energy-efficient convnets through approximate computing," in *WACV*, 2016.
[19] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," *arXiv preprint arXiv:1712.05877*, 2017.
[20] H. Gao *et al.*, "Ifq-net: Integrated fixed-point quantization networks for embedded vision," in *CVPR*, 2018.
[21] V. Sze *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.
[22] S. Gupta *et al.*, "Deep learning with limited numerical precision," in *ICML*, 2015.
[23] V. Peluso *et al.*, "Scalable-effort convnets for multilevel classification," in *ICCAD*. ACM, 2018, pp. 12:1–12:8.
[24] T. Na *et al.*, "Speeding up convolutional neural network training with dynamic precision scaling and flexible multiplier-accumulator," in *ISLPED*, 2016.
[25] L. Shan *et al.*, *A Dynamic Multi-precision Fixed-Point Data Quantization Strategy for Convolutional Neural Network*. Singapore: Springer Singapore, 2016, pp. 102–111.
[26] Compute library. [Online]. Available: https://developer.arm.com/technologies/compute-library
[27] D. Sun *et al.*, "Enabling embedded inference engine with ARM compute library: A case study," *CoRR*, vol. abs/1704.03751, 2017. [Online]. Available: http://arxiv.org/abs/1704.03751
[28] Linaro - leading software collaboration in the arm ecosystem. [Online]. Available: https://www.linaro.org/
[29] A. Geiger *et al.*, "Vision meets robotics: The kitti dataset," *IJRR*, 2013.
[30] D. Eigen *et al.*, "Depth map prediction from a single image using a multi-scale deep network," in *NIPS*, 2014.