

# Real-time single image depth perception in the wild with handheld devices

Filippo Aleotti, Giulio Zaccaroni, Luca Bartolomei, Matteo Poggi, Fabio Tosi, Stefano Mattoccia

**Abstract**—Depth perception is paramount to tackle real-world problems, ranging from autonomous driving to consumer applications. For the latter, depth estimation from a single image represents the most versatile solution, since a standard camera is available on almost any handheld device. Nonetheless, two main issues limit its practical deployment: i) the low reliability when deployed in-the-wild and ii) the demanding resource requirements to achieve real-time performance, often not compatible with such devices. Therefore, in this paper, we deeply investigate these issues showing how they are both addressable adopting appropriate network design and training strategies – also outlining how to map the resulting networks on handheld devices to achieve real-time performance. Our thorough evaluation highlights the ability of such fast networks to generalize well to new environments, a crucial feature required to tackle the extremely varied contexts faced in real applications. Indeed, to further support this evidence, we report experimental results concerning real-time depth-aware augmented reality and image blurring with smartphones in-the-wild.

**Index Terms**—Single image, monocular depth estimation, deep learning, mobile systems, smartphone.

## 1 INTRODUCTION

Depth perception is an essential step to tackle real-world problems such as robotics and autonomous driving, and some well-known sensors exist for this purpose. Among them, active sensing techniques such as Time-of-Flight (ToF) or LiDAR are often deployed in the application domains mentioned before. However, they struggle with typical consumer applications since ToF is mostly suited for indoor environments. At the same time, conventional LiDAR technology, frequently used for autonomous driving and other tasks, is too cumbersome and expensive for its deployment with relatively cheap and lightweight consumer handheld devices. Nonetheless, it is worth noting that active sensing technologies, mostly suited for indoor environments, are sometimes integrated into high-end devices as, for instance, occurs with the 2020 Apple iPad Pro.

Therefore, camera-based technologies are often the only viable strategies to infer depth in consumer applications, and among them, well-known methodologies are structured-light and stereo vision. The former typically requires an infrared camera and a specific pattern projector making it not suited for environments flooded by sunlight. The latter requires two appropriately spaced and synchronized cameras. Some recent high-end smartphones or tablets feature one or both technologies, although they are not yet widespread enough to be considered as standard equipment. Moreover, for stereo systems, the distance between the cameras (baseline) is necessarily narrow, limiting the depth range to a few meters away.

On the other hand, with the advent of deep-learning, recent years witnessed the rising of a further strategy to infer depth using a single standard camera available substantially in any consumer device. Compared to previous technologies for depth estimation, such an approach would enable to tackle all the limitations mentioned before. Nonetheless, single image depth estimation is seldom deployed in con-

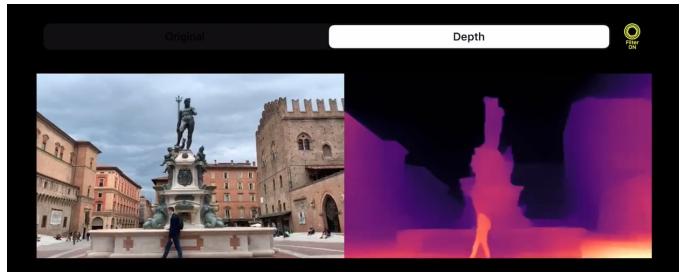


Fig. 1. **Depth perception in the wild with a mobile app.** Single image depth perception in the wild at nearly 60 FPS with an iPhone XS and the PyDNet [1] network.

sumer applications due to two main reasons. The first one concerns the low reliability of state-of-the-art methods when tackling depth estimation in the wild dealing with unpredictable environments not seen at training time, as necessarily occurs when targeting a massive amount of users facing heterogeneous environments. The second reason concerns the constrained computational resources available in handheld devices, such as smartphones or tablets, deployed for consumer applications. In fact, despite the steady progress in this field, the gap with system leveraging high-end GPUs is (and always will be) significant since power requirements heavily constrain handheld devices. Despite the limited computational resources available, most consumer applications need real-time performance. Arguing these facts, in this paper, we deeply investigate both the issues outlined so far. In particular, we will show how tackling them leveraging appropriate network design approaches, training strategies, and outlining how to map them on off-the-shelf consumer devices to achieve real-time performance, as shown in Figure 1. Indeed, our extensive evaluation highlights the ability of the resulting networks to robustly generalize to unseen environments, a crucial feature to tackle the heterogeneous contexts faced in real

consumer applications.

The paper is organized as follows. At first, we present previous works about monocular depth estimation on high performance and mobile devices, then we describe the framework that allows us to train a model, even a lightweight one, to be robust when deployed in the wild. We then evaluate a set of monocular networks already proposed in the literature on three benchmarks, deploying such models on mobile smartphones. Finally, we report how enabling real-time single-image depth estimation at the edge can be effectively exploited to tackle two well-known applications: depth aware augmented reality and blurring.

## 2 RELATED WORK

**Monocular depth estimation.** Even if depth estimation from multiple views has a long history in computer vision, depth from a single image [2], [3] started being considered feasible only with the advent of deep learning techniques. Indeed, obtaining depth values starting from a single image is an *ill-posed* problem, since an infinite number of real-world layouts may have generated the input image. However, learning-based methods, and in particular deep learning, proved to be adequate to face the problem. Initially were proposed supervised approaches [4], [5]. However, the need for ground truth measurements represents a severe restraint since an active sensor, as a LiDAR, together with specific manual post-processing is required to obtain such data. Therefore, effective deployment of supervised approaches is burdensome both in terms of time and costs. To overcome this limitation, solutions able to learn depth supervised only by images are incredibly appealing, and nowadays, several methods rely on simple stereo pairs or monocular video sequences for training. Among these, [6] is the first notable attempt leveraging stereo pairs, eventually improved exploiting traditional stereo algorithm [7], [8], visual odometry supervision [9], [10] or 3D movies [11]. On the other hand, methods leveraging monocular videos do not even require a stereo camera at training time, at the cost of learning depth estimation up to a scale factor. Starting with Zhou et al. [12], proposing to learn both depth and camera ego-motion, more recent methods propose to apply Direct Visual Odometry [13] or ICP [14] strategies to improve predictions. Additional cues have been used by more recent works, such as optical flow [15]–[18] or semantic segmentation [19]. Finally, it is worth noting that the two supervisions coming from stereo images and monocular sequences can be combined [20].

**Depth estimation on mobile systems.** Mobile devices are ubiquitous, and deep learning opened many applications scenarios [21]. Although sometimes server-side inference is unavoidable, maintaining the computation on-board is highly beneficial. It allows to get-rid of privacy issues, the need for tailored datacenters allowing to reduce costs as well to improve scalability. Moreover, although not critical for most consumer scenarios, full on-board processing does not dictate an internet connection. Despite limited computing capabilities, mostly constrained by power consumption issues and typically not comparable to those available in standard laptops or PCs, some authors proposed deep networks for depth estimation suited for mobile devices too. These works targeted stereo [22] and monocular [1], [23]

setups. Moreover, some authors proposed depth estimation architectures tailored for specific hardware setups, such as those based on dual-pixel sensors available in some recent Google's smartphones, as reported in [24], [25].

## 3 PROBLEMS AND REQUIREMENTS

The availability of more and more powerful devices paves the way for complex and immersive applications, in which users can interact with the nearby environment. As a notable example, augmented reality can be used to display interactive tools or concepts, avoiding to build a real prototype and thus cutting off costs. For this and many other applications, obtaining accurate depth information with a high frame rate is paramount to further enhance the interaction with the surrounding environment, even with devices devoid of active sensors. Almost any modern handheld device features at least a single camera and an integrated CPU within, typically, an ARM-based system-on-chip to cope with the constrained energy budget of such devices. Sometimes, especially in most new ones, a Neural Processing Unit (NPU) devoted to accelerating deep neural networks is also available. Inevitably, the resulting overall computation performance is far from conventional PC-based setups, and the availability of an NPU only partially fills this gap. Given these constraints, single-image depth perception would be rather appealing since it could seamlessly deal with dynamic contexts, whereas other techniques such as *structure from motion* (SfM) would struggle. However, these techniques are computationally demanding, and most state of the art approaches would not fit the computational resources available in handheld devices. Moreover, regardless of the computing requirements, training the networks for predictable target environments is not feasible for consumer applications. Thus the depth estimation network shall be robust to any faced deployment scenarios and possibly invariant to the training data distribution. A client-server approach would soften some of the computational issues, although with notable disadvantages the need for an internet connection and a poorly scaling of the whole overall system when the number of users increases.

To get rid of all the issues mentioned above and to deal with practical applications, we will describe next how to achieve real-time and robust single image depth perception on low-power architectures found in off-the-shelf handheld devices.

## 4 FRAMEWORK OVERVIEW

In this section, we introduce our framework aimed at enabling single image depth estimation in the wild with mobile devices, devoting specific attention to iOS and Android systems. Before the actual deployment on the target handheld device, our strategy requires an offline training procedure typically carried out on power unconstrained devices such as a PC equipped with a high-end GPU. We will discuss in the reminder the training methodology, leveraging *knowledge distillation*, deployed to achieve our goal in a limited amount of time and the dataset adopted for this purpose. Another critical component of our framework is a lightweight network enabling real-time processing on the target handheld devices. Purposely, we will introduce

and thoroughly assess the performance of state of the art networks fitting this constraint.

#### 4.1 Off-line training

As for most learning-based monocular depth estimation models, our proposal is trained off-line on standard workstations, equipped with one or more GPUs, or through cloud processing services. In principle, depending on the training data available, one can leverage different training strategies: supervised, semi-supervised or self-supervised training paradigms. Moreover, as done in this paper, cheaper and better-scaling supervision can be conveniently obtained from another network, by leveraging knowledge distillation to avoid the need for expensive ground truth labels, through a teacher-student network.

When a large enough dataset providing ground truth labels inferred by an active sensor is available, such as [26], [27], (semi-)supervised fashion is certainly valuable since it enables, among other things, to disambiguate difficult regions (e.g. texture-less regions such as walls). Unfortunately, large datasets with depth labels are not available or extremely costly and cumbersome to obtain. Therefore, when this condition is not met, self-supervised paradigms enable to train with (potentially) countless examples, at the cost of a more challenging training setup and typically less accurate results. Note that, depending on the dataset, a strong depth prior can be distilled even if are not available depth labels provided by an active sensor. For instance, [7], [8] exploit depth values from a stereo algorithm, while [28] relies on a SfM pipeline. Finally, supervision can be distilled from other networks as well, for the stereo [29] and monocular [30] setup. The latter is the strategy followed in this paper. Specifically, we use as teacher the *MiDaS* network proposed in [11]. This strategy allows us to speed-up the training procedure of the considered lightweight networks significantly, since doing this from scratch according to the methodology proposed in [11] would take much much longer time (weeks vs days) since mostly bounded by proxy labels generation. Moreover, it is worth noting that given a reliable teacher network, pre-trained in a semi or self-supervised manner, such as [11], it is straightforward to distill an appropriate training dataset since any collection of images is potentially suited to this aim. We will describe next the training dataset used for our experiments made of a bunch of single images belonging to well-known popular datasets.

#### 4.2 On-device deployment and inference

Once outlined the training paradigm, the next issue concerns the choice of a network capable of learning from the teacher how to infer meaningful depth maps and, at the same time, able to run in real-time on the target handheld devices. Unfortunately, only a few networks described next potentially fulfil these requirements, in particular, considering the ability to run in real-time on embedded systems.

Once identified and trained a suitable network, its mapping on a mobile device is nowadays quite easy. In fact, there exist various tools that, starting from a deep learning framework as PyTorch [31] or TensorFlow [32], can export, optimize (e.g. perform weights quantization) and execute models even leveraging mobile GPU [33] on principal operating

systems (OS). In some cases, the target OS exposes utilities and tools to improve the performances further. For instance, starting from iOS 13, neural networks deployed on iPhones can use the GPU or even the Apple Neural Engine (ANE) thanks to Metal and Metal Performance Shaders (MPS), thus largely improving the runtime performances. We will discuss in the next section how to map the networks on iOS and Android devices using TensorFlow and PyTorch as high-level development frameworks.

## 5 LIGHTWEIGHT NETWORKS FOR SINGLE IMAGE DEPTH ESTIMATION

According to the previous discussion, only a subset of the state of the art single image depth estimation networks fits our purposes. Specifically, we consider the following publicly available lightweight architectures: PyDNet [1], DSNet [19] and FastDepth [23]. Moreover, we also include a representative example of a large state of the art network MonoDepth2, proposed in [20]. It is worth to notice that other and more complex state-of-the-art networks, as [7], could be deployed in place within the proposed framework. However, this might come at the cost of higher execution time on the embedded device and, potentially, overhead for the developer in case of custom layers not directly supported by the mobile executor (e.g., the correlation layer used in [7]).

**MonoDepth2.** An architecture deploying a ResNet encoder, proposed initially in [34], made of 18 feature extraction layers, shrinking the input by a factor of  $\frac{1}{32}$ . Then, the dense layers are replaced in favour of a decoder module, able to restore the original input resolution and output an estimated depth map. At each level in the decoder,  $3 \times 3$  convolutions with skip connections are performed, followed by a  $3 \times 3$  convolution layer in charge of depth estimation. The resulting network can predict depths at different scales, counting 14.84 M parameters. It is worth to notice that in our evaluation we do not rely on ImageNet [35] pre-training for the encoder for fairness to other architectures not pre-trained at all.

**PyDNet.** This network, proposed in [1], features a pyramidal encoder-decoder design able to infer depth maps from a single RGB image. Thanks to its small size and design choices, PyDNet can run on almost any device including low-power embedded platforms [36], such as the Raspberry Pi 3. In particular, the network exploits 6 layers to reduce the input resolution at  $\frac{1}{64}$ , restored in the depth domain by 5 layers in the decoder. Each layer in the decoder applies  $3 \times 3$  convolutions with 96, 64, 32, 8 feature channels, followed by a  $3 \times 3$  convolution in charge of depth estimation. Notice that, to keep low the resources and inference time, top prediction of PyDNet is at half resolution, so the final depth map is obtained through an upsampling operation. We adopt the mobile implementation provided by the authors, publicly available online<sup>1</sup>, which differs from the paper network by small changes (e.g. transposed convolutions have been replaced by upsampling and convolution blocks). The network counts 1.97 M parameters.

1. <https://github.com/FilippoAleotti/mobilePydnet>

**FastDepth.** Proposed by Wofk et al. [23], this network can infer depth predictions at 178 fps with an NVIDIA Jetson TX2 GPU. This notable speed is the result of design choices and optimization steps. Specifically, the encoder is a MobileNet [21], thus suited for execution on embedded devices. The decoder consists of 6 layers, each one with a depth-wise separable convolution, with skip connections starting from the encoder (in this case, features are combined with addition). However, it is worth observing that the highest frame rate previously reported is achievable only exploiting both pruning [37] and hardware-specific optimization techniques. In this paper, we do not rely on such strategies for fairness with other networks. The network counts 3.93 M parameters.

**DSNet** This architecture is part of  $\Omega$ Net [19], an ensemble of networks predicting not only the depth of the scene starting from a single view but also the semantic segmentation, camera intrinsic parameters and if two frames are provided, the optical flow. In our evaluation we consider only the depth estimation network DSNet, inspired by PyDNet, which contains a feature extractor able to decrease the resolution by  $\frac{1}{32}$ , followed by 5 decoding layers able to infer depth predictions starting from the current features and previous depth estimate. In the original architecture, the last decoder also predicts per-pixel semantic labels through a dedicated layer, removed in this work. With this change, the network counts 1.91 M of parameters, 0.2 M fewer than the original model.

## 6 DATASETS

In our evaluation, we use four datasets. At first, we rely on the KITTI dataset to assess the performance of the four networks when trained with the standard self-supervised paradigm deployed typically in this field [20]. Then, we retrain from scratch the four networks using the paradigm previously outlined, distilling proxy labels by employing the pre-trained MiDaS network [11] made available by the same authors. For this task, we use a novel dataset, referred to as WILD, described next. We then evaluate the networks trained according to this methodology on the TUM RGBD [38] and NYUv2 [39] dataset to assess their generalization capability.

**KITTI.** The KITTI dataset [40] contains 61 scenes collected by moving car equipped with a LiDAR sensor and a stereo rig. Following [20], we select a split of 697 images for testing, while 39810 and 4424 images are used respectively for preliminary training and validation purpose. Moreover, we use it to assess the generalization capability of the networks in the wild during the second part of our evaluation.

**WILD** The Wild dataset (W), introduced in this paper, consists of a mixture of Microsoft COCO [41] and OpenImages [42] datasets. Both datasets contain a large number of internet photos, and they do not provide depth labels. Moreover, since video sequences nor stereo pairs are available, they are not suited for conventional self-supervised guidance methods (e.g. SfM or stereo algorithms). On the other hand, they cover a broad spectrum of various real-world situations, allowing to face both indoor and outdoor environments, deal with everyday objects and various depth ranges. We

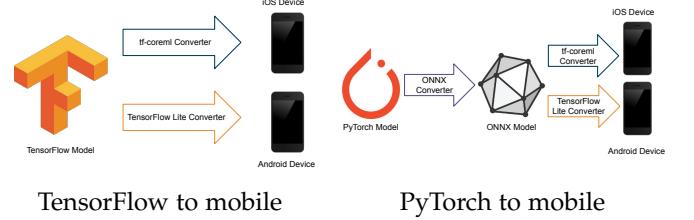


Fig. 2. **Porting a network from a standard ML frameworks to a mobile device.** Both TensorFlow and PyTorch models can be easily converted into models suited for mobile execution using libraries available for each OS.

select almost 447000 frames for training purposes<sup>2</sup>. Then, we distilled the supervision required by our networks with the robust monocular architecture proposed in [11] with the weights publicly available. We point out once again that our supervision protocol has been carefully chosen mostly for practical reasons. It takes a few days to distill the WILD dataset by running MiDaS (using the publicly available checkpoints) on a single machine. On the contrary, to obtain the same data used to train the network as in [11], it would require an extremely intensive effort. Doing so, we can scale better: since we trust the teacher, we could, in principle, source knowledge from various and heterogeneous domains on the fly. Of course, the major drawbacks of this approach are evident: we need an already available and reliable teacher, and the accuracy of the student is bounded to the one of the teacher. However, we point out that the training scheme proposed in [11] is general, so it can also be applied in our case, and that we already expect a margin with state-of-the-art networks due to the lightweight size of mobile architectures considered. For these reasons, we believe that our approach is beneficial to source a fast prototype than can be improved later leveraging other techniques if needed. This belief is supported by experimental results presented later in the paper.

**TUM RGBD.** The TUM RGBD (3D Object Reconstruction category) dataset [38] contains indoor sequences framing people and furniture. We adopt the same split of 1815 images used in [28] for evaluation purposes only.

**NYUv2.** The NYUv2 dataset [39] is an indoor RGBD dataset acquired with a Microsoft Kinect device. It provides more than 400k raw depth frames and 1449 densely labelled frames. As for the previous dataset, we adopt the official test split containing 649 images for generalization tests.

## 7 MAPPING ON MOBILE DEVICES

Since we aim at mapping single image depth estimation networks on handheld devices, we briefly outline here the steps required to carry out this task.

As depicted in Figure 2, different tools are available according to both the deep learning framework and the target OS. With TensorFlow models, weights are processed using *tf-coreml* converter in case of iOS deployment or *TensorFlow Lite* converter when targeting an Android device. On the other hand, when starting from PyTorch models, a further intermediate step is required. In particular, stored weights

2. Details concerning the WILD dataset are available at this link: <https://github.com/FilippoAleotti/mobilePydnet>

Network	Lower is better				Higher is better		
	Abs Rel	Sq Rel	RMSE	log RMSE	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
PyDNet	0.153	1.363	6.030	0.252	0.789	0.918	0.963
FastDepth	-	-	-	-	-	-	-
DSNet	0.130	0.909	5.022	0.207	0.842	0.948	0.979
MonoDepth2 †	<b>0.132</b>	<b>1.044</b>	<b>5.142</b>	<b>0.210</b>	<b>0.845</b>	<b>0.948</b>	<b>0.977</b>
PyDNet†	0.154	1.307	5.556	0.229	0.812	0.932	0.970
FastDepth†	0.156	1.260	5.628	0.231	0.801	0.930	0.971
DSNet†	0.159	1.272	5.593	0.233	0.800	0.932	0.971

TABLE 1

**Quantitative results on Eigen split.** † indicates models trained according to [20] training framework, otherwise we report results provided in each original paper.



Fig. 3. **Qualitative results on KITTI.** All the models have been trained equally using the framework of [20] on the Eigen split of KITTI.

are converted in the Open Neural Network Exchange format (ONNX), a common representation allowing for porting architectures implemented in a source framework into a different target framework. This conversion is possible seamlessly if the networks consist of standard layers, while is not straightforward in case of custom modules (e.g., correlation layers as in monoResMatch [7]). Although these tools typically enable to perform weights quantization during the model conversion phase to the target environment, we refrain from applying quantization to maintain the original accuracy of each network. In the experimental results section, we will provide execution time mapping the networks on mobile devices following the porting strategy outlined so far.

## 8 EXPERIMENTAL RESULTS

In this section, we thoroughly assess the performance of the considered networks with standard datasets deployed in this field. At first, since differently from other methods FastDepth [23] was not initially evaluated on KITTI, we carry out a preliminary evaluation of all networks on such dataset. Then, we train from scratch the considered networks according to the framework outlined on the Wild dataset, evaluating their generalization ability. Finally, we show how to take advantage of the depth maps inferred by such networks for two applications particularly relevant for mobile devices.

### 8.1 Evaluation on KITTI

At first, we first investigate the accuracy of the considered networks on the KITTI dataset. Since the models have been developed with different frameworks (PyDNet in TensorFlow, the other two in PyTorch) and trained on different datasets (FastDepth on NYU v2 [39], others on the Eigen [4] split KITTI [40]), we implement all the networks in PyTorch. This strategy allows us to adopt the same self-supervised protocol proposed in [20] to train all the models. This choice is suited for the KITTI dataset since it exploits stereo sequences enabling to achieve the best accuracy. Given two images  $I$  and  $I^\dagger$ , with known intrinsic parameters ( $K$  and  $K^\dagger$ ) and relative pose of the cameras ( $R, T$ ), the network

predicts depth  $\mathcal{D}$  allowing to reconstruct the reference image  $I$  from  $I^\dagger$ , so:

$$\hat{I} = \omega(I^\dagger, K^\dagger, R, T, K, \mathcal{D}) \quad (1)$$

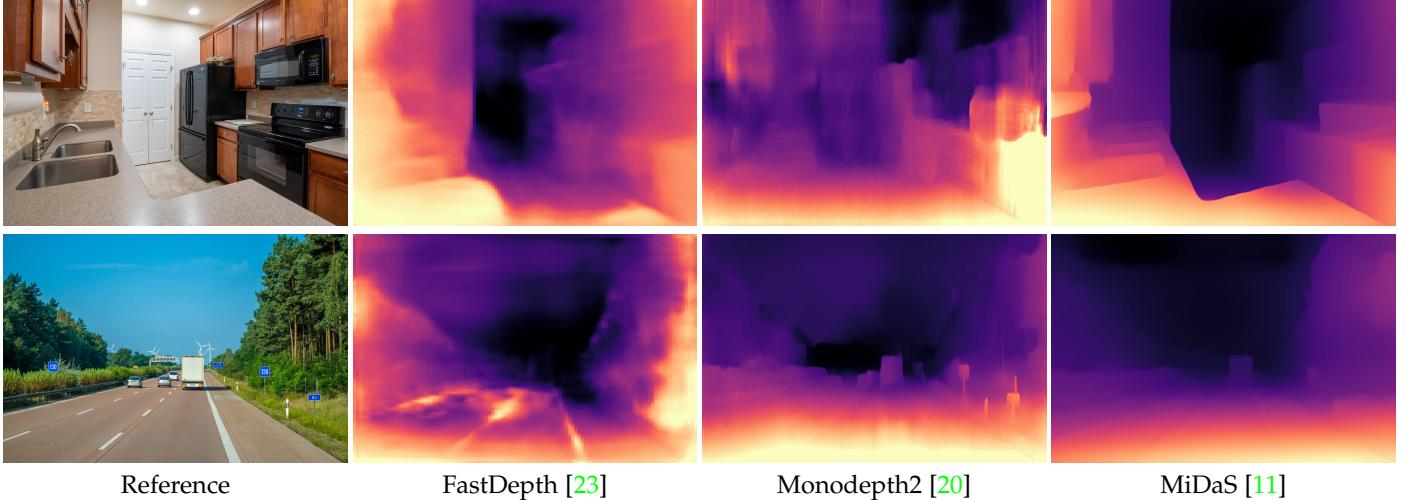
where  $\omega$  is a differentiable warping function. Then, the difference between  $\hat{I}$  and  $I$  can be used to supervise the network, thus improving  $\mathcal{D}$ , without any ground truth. The loss function used in [20] is composed by a photometric error term  $p_e$  2 and an edge-aware regularization term  $L_s$ .

$$p_e(I, \hat{I}) = \alpha \frac{(1 - \text{SSIM}(I, \hat{I}))}{2} + (1 - \alpha) \|I - \hat{I}\|_1 \quad (2)$$

$$\mathcal{L}_s = \|\delta_x \mathcal{D}_t^*\|_1 e^{-\|\delta_x I_t\|_1} + \|\delta_y \mathcal{D}_t^*\|_1 e^{-\|\delta_y I_t\|_1} \quad (3)$$

where SSIM is the structure similarity index [43], while  $\mathcal{D}^* = \mathcal{D}/\overline{\mathcal{D}}$  is the mean normalized inverse depth proposed in [44]. We adopt the M configuration of [20] to train all the models. Doing so, given the reference image  $I_t$ , at training time we also need  $\{I_{t-1}, I_{t+1}\}$ , that are respectively the previous and the next frames in the sequence, to leverage the supervision from monocular sequences as well. Purposefully, a pose network is trained to estimate relative poses between the frames in the sequence as in [20]. Moreover, *per-pixel minimum* and *automask* strategies are used to preserve sharp details: the former select best  $p_e$  among multiple views according to occlusions, while the latter helps to filter out pixels that do not change between frames (e.g. scenes with a non-moving camera or dynamic objects that are moving at the same speed of the camera), thus breaking the *moving camera in a stationary world* assumption (more details are provided in the original paper [20]). Finally, intermediate predictions, when available, are upsampled and optimized at input resolution.

Considering that all the models have been trained with different configurations on different datasets, we re-train all the architectures exploiting the training framework of [20] for a fair comparison. Specifically, we run 20 epochs of training for each model, decimating the learning rate after 15, on the Eigen train split of KITTI. We use Adam optimizer



**Fig. 4. Predictions in the wild.** We provide qualitative results for indoor and outdoor internet images. For each network, we used the checkpoints publicly available. It can be noticed how the networks trained on a single dataset, both in a supervised (FastDepth) and self-supervised (Monodepth2), are not able to generalize well on a different setup. On the contrary, the network trained on various datasets (MiDaS) produces better results. Images from Pexels <https://www.pexels.com/>.

[45], with an initial learning rate of  $10^{-4}$ , and minimize the highest three available scales for all the network except FastDepth, which provides full-resolution (i.e.  $640 \times 192$ ) predictions only. Since the training framework expects a normalized inverse depth as the output of the network, we replace the last activation of each architecture (if present) with a sigmoid.

Table 1 summarizes the experimental results of the models tested on the Eigen split of KITTI. The top four rows report the results, if available, provided in the original papers, while last three the accuracy of models re-trained within the framework described so far. This test allows for evaluating the potential of each architecture in fair conditions, regardless of the specific practices, advanced tricks or pre-training being deployed in the original works. Not surprisingly, larger MonoDepth2 model performs better than the three lightweight models, showing non-negligible margins on each evaluation metric when trained in fair conditions. Among these latter, although their performance is comparable, PyDNet results more effective with respect to FastDepth and DSNet on most metrics, such as RMSE and  $\delta < 1.25$ .

Figure 3 shows some qualitative results, enabling us to compare depth maps estimated by the four networks considered in our evaluation on a single image from the Eigen test split.

## 8.2 Evaluation in the wild

In the previous section, we have assessed the performance of the considered lightweight networks on a data distribution similar to the training one. Unfortunately, this circumstance is seldom found in most practical applications, and typically it is not known in advance where a network will be deployed. Therefore, how to achieve reliable depth maps in the wild? In Figure 4 we report some qualitative results about original pre-trained networks on different scenarios. Notice that the first two networks have strong constraints about input size ( $224 \times 224$  for [23],  $1024 \times 320$  for [20]) that these networks internally apply, imposed by how these

models have been trained in their original context. Although this limitation, FastDepth (second column) can predict a meaningful result in an indoor environment (first row), not in outdoor (second row). It is not surprising since the network was trained on NYUv2, which is an indoor dataset. Monodepth2 [20] suffers from the same problem, highlighting that this issue is not concerned with the network size (smaller the first, larger the second) or training approach (supervised the first, self-supervised the second), but it is rather related to the training data. Conversely, MiDaS by Ranftl et al. [11], is effective in both situations. Such robustness comes from a mixture of datasets, collecting about 2M frames covering many different scenarios, used to train a large ( $\sim 105$ M parameters) and very accurate monocular network.

We leverage this latter model to *distill* knowledge and train the lightweight models compatible with mobile devices. As mentioned before, this strategy allows use to *use* MiDaS knowledge for faster training data generation compared to time-consuming pipelines used to train it, such as COLMAP [47], [48]. Moreover, it allows us to generate additional training samples and thus a much more scalable training set, potentially from any (single) image. Therefore, in order to train our network using the WILD dataset, we first generate proxy labels with MiDaS for each training image of this dataset. Then, obtained such proxy labels, we train the networks using the following loss function:

$$\mathcal{L}(D_x^s, D_{gt}) = \alpha_l \| (D_x^s - D_{gt}) \| + \alpha_s \mathcal{L}_g(D_x^s, D_{gt}) \quad (4)$$

where  $\mathcal{L}_g$  is the gradient loss term defined in [28],  $D_x^s$  the predictions of the network at scale  $s$  (bilinearly upsampled to full resolution) and  $D_{gt}$  is the proxy depth. The weight  $\alpha_s$  depends on the scale  $s$  and is halved at each lower scale. On the contrary,  $\alpha_l$  is fixed and set to 1. Intuitively, the  $L^1$  norm penalizes differences w.r.t proxies, while  $\mathcal{L}_g$  helps to preserve sharp edges. We train the models for 40 epochs, halving the learning rate after 20 and 30, with a batch size of 12 images, with an input size of  $640 \times 320$ . We set the initial

Network	Dataset	Lower is better			Higher is better		
		Abs Rel	Sq Rel	RMSE	log RMSE	$\delta < 1.25$	$\delta < 1.25^2$
Ranftl et al. [11]	TUM	<b>0.125</b>	<b>0.148</b>	0.832	0.195	<b>0.857</b>	<b>0.944</b>
Li et al. [28]	TUM	0.135	0.158	0.852	0.209	0.826	0.942
MonoDepth2	TUM	<b>0.147</b>	<b>0.180</b>	<b>0.916</b>	<b>0.222</b>	<b>0.811</b>	<b>0.927</b>
PyDNet	TUM	0.166	0.210	0.978	0.244	0.767	0.921
DSNet	TUM	0.168	0.215	0.994	0.248	0.762	0.917
FastDepth	TUM	0.160	0.209	0.982	0.241	0.780	0.918
Ranftl et al. [11]	KITTI	<b>0.157</b>	<b>1.144</b>	<b>5.672</b>	<b>0.225</b>	<b>0.780</b>	<b>0.942</b>
Li et al. [28]	KITTI	0.227	2.081	7.841	0.325	0.621	0.854
MonoDepth2	KITTI	0.164	<b>1.194</b>	6.000	<b>0.239</b>	0.752	0.928
PyDNet	KITTI	<b>0.162</b>	1.272	6.138	<b>0.239</b>	<b>0.760</b>	0.927
DSNet	KITTI	0.164	1.203	<b>5.977</b>	<b>0.239</b>	0.754	<b>0.929</b>
FastDepth	KITTI	0.168	1.227	6.017	0.241	0.741	0.927
Ranftl et al. [11]	NYU	<b>0.100</b>	<b>0.061</b>	<b>0.407</b>	<b>0.132</b>	<b>0.905</b>	<b>0.984</b>
Li et al. [28]	NYU	0.149	0.116	0.560	0.189	0.782	0.958
MonoDepth2	NYU	<b>0.123</b>	<b>0.082</b>	<b>0.473</b>	<b>0.160</b>	<b>0.848</b>	<b>0.974</b>
PyDNet	NYU	0.130	0.091	0.493	0.168	0.827	0.969
DSNet	NYU	0.134	0.096	0.505	0.171	0.820	0.968
FastDepth	NYU	0.129	0.090	0.492	0.167	0.833	0.971
							0.994

TABLE 2

**Evaluation of generalization capability.** The three groups from top to bottom report experimental results concerned, respectively, with (top) TUM dataset, (middle) KITTI Eigen and (bottom) NYUV2.

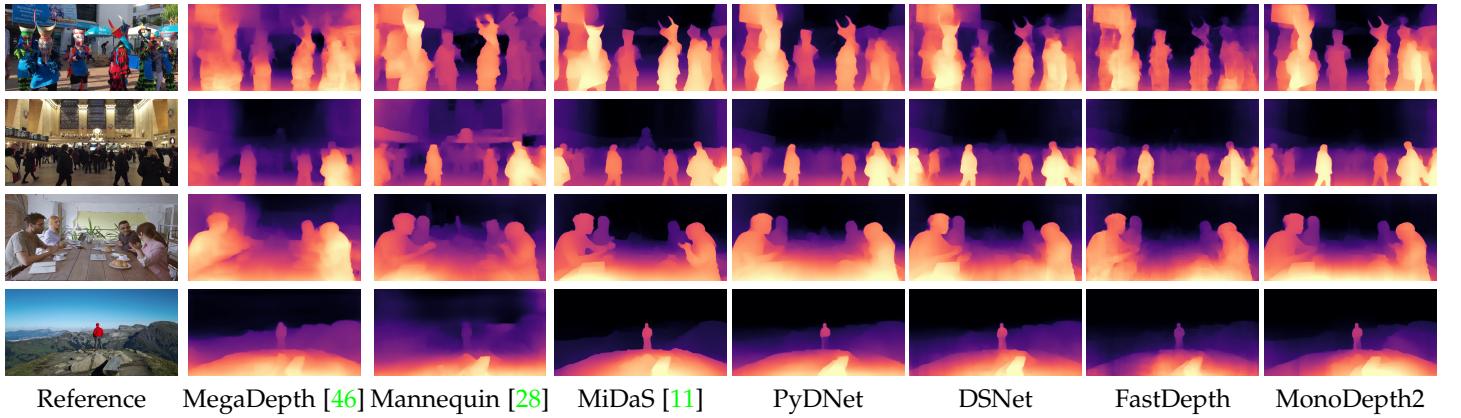


Fig. 5. **Qualitative results from internet photos.** From left to right, the reference image from Pexels website, depths from [46], [11] and our prediction, respectively for PyDNet and DSNet.

value of  $\alpha_s$  to 0.5 for all networks except for FastDepth, set to 0.01. Additionally, for MonoDepth2 and FastDepth feature upsampling through the nearest neighbour operator in the decoder phase have been replaced with bilinear interpolation. These changes were necessary to mitigate some checkboard artefacts found in depth estimations inferred by these networks following the training procedure outlined.

Table 2 collects quantitative results on three datasets, respectively TUM [38] (3D object reconstruction category), KITTI Eigen split [4] and NYU [39]. On each dataset, we first show the results achieved by large and complex networks MiDaS [11] and the model by Li et al. [28] (using the single frame version), both trained in the wild on a large variety of data. The table also reports results achieved by the four networks considered in our work trained on the WILD dataset exploiting knowledge distillation from MiDaS. First and foremost, we highlight how MiDaS performs in general better than [28], emphasizing the reason to distil knowledge from it.

Considering lightweight compact models PyDNet, DSNet and FastDepth we can notice that the margin between them and MiDaS is often non-negligible. Similar behaviour occurs for the significantly more complex network MonoDepth2

despite in general more accurate than other more compact networks, except on KITTI where it turns out less accurate when trained in the wild. However, considering the massive gap in terms of computational efficiency between compact networks and MiDaS analyzed later, that makes MiDaS not suited at all for real-time inference on the target devices the outcome reported in Table 2 is not so surprising. Looking more in details the outcome of lightweight networks, PyDNet is the best model on KITTI when trained in the wild and also achieves the second-best accuracy on NYU, with minor drops on TUM. Finally, DSNet and FastDepth achieve average performance in general, never resulting in the best on any dataset.

Figure 5 shows some qualitative examples of depth maps processed from internet pictures by MegaDepth [46], the model by Li et al. [28], MiDaS [11] and the fast networks trained through knowledge distillation in this work.

Finally, in Figure 6 we report some example of failure cases of MiDaS (in the middle column) inherited by student networks. Since both networks fail, the problem is not attributable to their different architecture. Observing the figure, we can notice that such behavior occurs in very ambiguous scenes such as when dealing with mirrors or flat



Fig. 6. **Failure cases.** Example of failure cases of single image depth estimation. From left to right: input image, depth predicted by the teacher and by the student.

Network	MAC (G)	FPS
MiDaS	172.4	0.20
MonoDepth2	16.01	9.94
DSNet	9.48	11.05
PyDNet	9.25	58.86
FastDepth	3.61	50.31

TABLE 3

**Performance on smartphones.** We measure both the number of *multiply-accumulate* operation (MAC) and the FPS of monocular networks on an iPhone XS, using an input size of  $640 \times 384$ , averaged on 50 inferences.

surfaces with content aimed at inducing optical illusions in the observers.

### 8.3 Performance analysis on mobile devices

After training the considered architectures on the WILD dataset, the stored weights can be converted into mobile-friendly models using tools provided by deep learning frameworks. Moreover, as previously specified, in our experiments, we perform only model conversion avoiding weights quantization not to alter the accuracy of the original network.

Table 3 collects stats about the considered networks. Specifically, we report the number of *multiply-accumulate* operations (MAC), computed with TensorFlow utilities, and the frame rate (FPS) measured when deploying the converted models on an Apple iPhone XS. Measurements are gathered processing  $640 \times 384$  images and averaging over 50 consecutive inferences. On top, we report the performance achieved by MiDaS, showing that it requires about 5 seconds on a smartphone to process a single depth map, performing about 170 billion operations. This evidence highlights how, despite being much more accurate, as shown before, this vast network is not suited at all for real-time processing on mobile devices. Moving on more compact models, we can notice how MonoDepth2 reaches nearly 10 FPS performing one order of magnitude fewer operations. DSNet and PyDNet both perform about 9 billion operations, but the latter allows for much faster inference, close to 60 FPS and about 6 times faster than previous models. Since the number of operations is almost the same for DSNet and PyDNet, we reconduct this performance discrepancy to low-level optimization of some specific modules. Finally, FastDepth performs 3 times fewer operations, yet runs slightly slower than PyDNet when deployed with the same degree of optimization of the other networks on the iPhone XS.

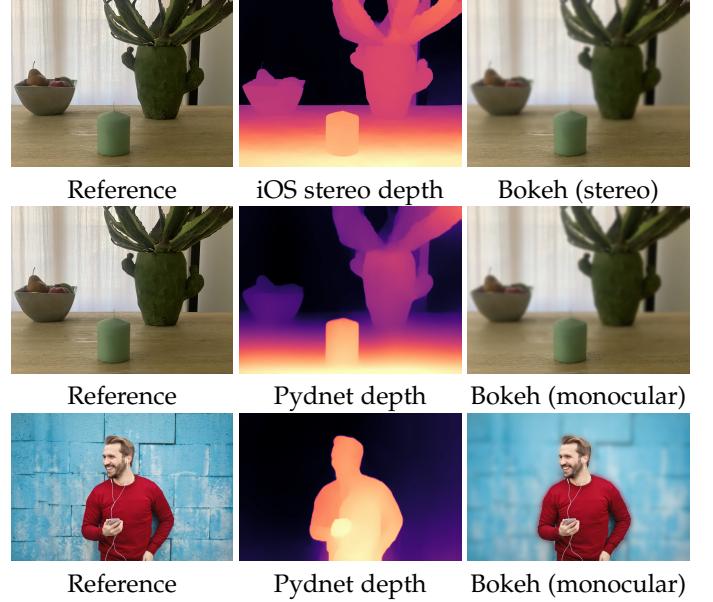


Fig. 7. **Bokeh effect.** Given the reference image acquired with an iPhone XS, we smooth farther pixels in the image using depth values provided by the native stereo method (first row) and PyDNet monocular network (second row), obtaining similar results. Moreover, we can apply the same effect with PyDNet using images from the web (third row).

Summarizing the performance analysis reported in this section and the previous accuracy assessment concerning the deployment of single image depth estimation in the wild, our experiments highlight PyDNet as the best trade-off between accuracy and speed when targeting embedded devices.

A video showing the deployment of PyDNet with an iPhone XS framing an urban environment is available at [youtube.com/watch?v=LRfGabIYZNw](https://youtube.com/watch?v=LRfGabIYZNw).

At the following link is also available a PyDNet web demo with client-side inference carried out by TensorFlow JS: [filippoaleotti.github.io/demo\\_live](http://filippoaleotti.github.io/demo_live).

## 9 APPLICATIONS OF SINGLE IMAGE DEPTH ESTIMATION

Once exhaustively assessed the performance of the considered lightweight networks, we present two well-known applications that can significantly take advantage of real-time and accurate single image depth estimation. For these experiments, we use the PyDNet model trained on the WILD dataset, as described in previous sections.

**Bokeh effect.** The first application consists of a *bokeh filter*, aimed at blurring an image according to the distance from the camera. More precisely, in our implementation, given a threshold  $\tau$ , all the pixels with a relative inverse depth larger than  $\tau$  are blurred by a  $25 \times 25$  Gaussian kernel.

For our experiments, we captured a stereo pair using the rear cameras of an iPhone XS, and then using its API, we inferred a depth map to obtain a baseline. We also fed the PyDNet network the single reference image of the stereo pair. Figure 7 depicts the depth maps inferred by the stereo and monocular approach and the outcome of the bokeh filter. From the figure, we can notice that even if the depth map inferred by the monocular system is not in

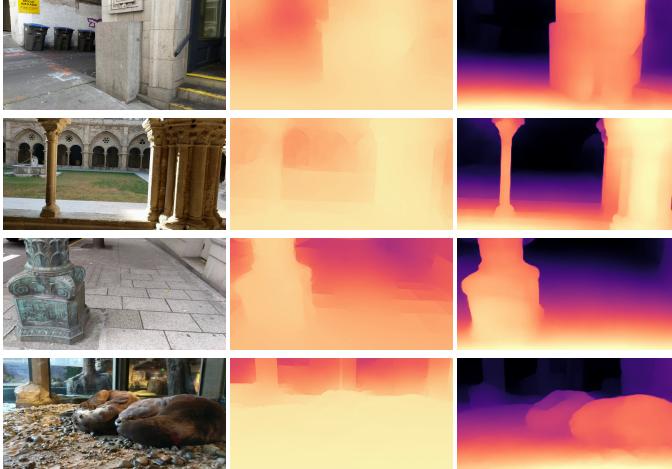


Fig. 8. **Qualitative comparison with other occlusion-aware AR methods.** From left to right, the input image, the depth from [49] and PyDNet predictions.

scale as the stereo one, it preserves pretty well details and also allows to retrieve the relative distance of objects. This latter feature, combined with the need for a single image is highly desirable in many consumer applications like the one described. Additionally – since the distance between the two imaging sensors of a stereo setup of a mobile phone is short– the parallax effect enabling to infer depth with stereo vanishes close to the camera. On the contrary, a monocular system is agnostic to this problem. For this experiment, we set  $\tau$  equals to 0.9 and 0.7 for stereo and monocular depth, respectively.

Finally, another advantage consists of enabling the bokeh effect even when a stereo pair is not available, for instance, dealing with images sampled from the web (third row in Figure 7). From it, we can notice how processing a single input image enables a charming effect. Notice that bokeh effect with stereo is not applicable in this case since the stereo pair is not available as frequently occurs in practice.

**Augmented reality with depth-aware occlusion handling.** Modern augmented reality (AR) frameworks for smartphones allow robust and consistent integration of virtual objects on flat areas leveraging camera tracking. However, they miserably fail when the scene contains occluding objects protruding from the flat surfaces. Therefore, in AR scenarios, dense depth estimation is paramount to handle properly physical interactions with the real world, such as occlusions. Unfortunately, most methods rely only on sparse depth measurements for a few points in the scene appropriately scaled exploiting the sensor suite of modern smartphones comprising accelerometers, gyroscope, etcetera. Although some authors proposed to densify such sparse measurements, it is worth observing that dynamic objects in the sensed scene may yield incorrect, sparse estimation and thus these methods need to filter out moving points [49]. We argue that single image depth estimation may enable full perception of the scene suited for many real-world use cases potentially avoiding at all the issues outlined so far. The only remaining issue, concerned with the unknown scale factor intrinsic in a monocular system can be robustly addressed

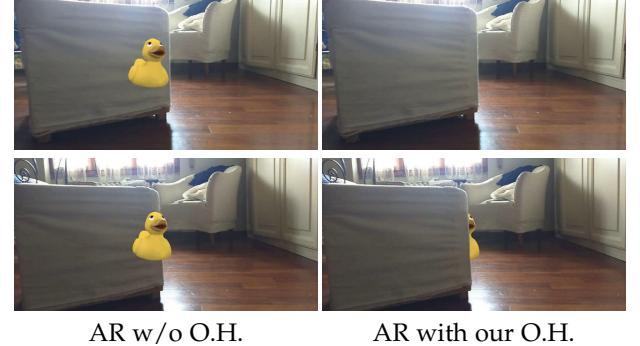


Fig. 9. **AR with occlusion handling (O.H.).** On the left, vanilla AR enabled by an Android device with ARCore. On the right, instead, our depth-aware AR enabled by single image depth prediction with PyDNet for occlusion handling.

leveraging, as described next, one of the multiple sparse depth measurements in scale among those made available by standard AR frameworks.

Purposely, we developed a mobile application capable of handling in real-time object occlusions by combining AR frameworks, such as ARCore or ARKit, and a robust and lightweight monocular depth estimation network. To achieve this goal, we first exploit the AR framework to retrieve low-level information, such as the pose of the camera and the position of anchors in the sensed environment. Then, we retrieve depth for anchor points and, by comparing such measurements with monocular depth predictions, we can prevent to render occluded regions. At each frame, the scale factor issue is tackled within a robust RANSAC-based framework fed with the sparse and potentially noisy depth measurements provided by the AR framework and the dense depth map estimated by the monocular network. Differently from other approaches, such as [49] and [50], our networks do not require SLAM points to infer dense depth maps nor a fine-tuning of the network on the input video data. In our case, a single image and at least one point in scale suffice to obtain absolute depth perception. Consequently, we do not rely on other techniques (e.g. optical flow or edge localization) in our whole pipeline for AR. Nevertheless, it can be noticed in Figure 8 how our strategy coupled with PyDNet can produce competitive and detailed depth maps leveraging a single RGB image only. Figure 9 shows some qualitative examples of an AR application, i.e. visualization of a *virtual duck* in the observed scene. Once positioned on a surface, we can notice how foreground elements do not correctly hide it without proper occlusion handling. In contrast, our strategy allows for a more realistic experience, thanks to the dense and robust depth map inferred by PyDNet and sparse anchors provided by the AR framework.

## 10 CONCLUSION

In this paper, we proposed a strategy to train single image depth estimation networks, focusing our attention on lightweight ones suited for handheld devices characterized by severe constraints concerning power consumption and computational resources. An exhaustive evaluation highlights that real-time depth estimation from a single image

in the wild is feasible by adopting appropriate network design and training strategies. By distilling knowledge from complex architecture, not suited for mobile deployment, we have shown that it is possible to develop accurate yet fast networks enabling for a variety of AR applications on consumer smartphones. We also reported the effectiveness of such an approach in two notable application scenarios concerning depth aware blurring and augmented reality.

## ACKNOWLEDGEMENT

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

## REFERENCES

- [1] M. Poggi, F. Aleotti, F. Tosi, and S. Mattoccia, "Towards real-time unsupervised monocular depth estimation on cpu," in *IROS*, 2018.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *3DV*, 2016.
- [6] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *CVPR*, 2017.
- [7] 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)*, June 2019.
- [8] J. Watson, M. Firman, G. J. Brostow, and D. Turmukhambetov, "Self-supervised monocular depth hints," in *ICCV*, 2019.
- [9] N. Yang, R. Wang, J. Stuckler, and D. Cremers, "Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 817–833.
- [10] 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.
- [11] K. Lasinger, R. Ranftl, K. Schindler, and V. Koltun, "Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer," *arXiv preprint arXiv:1907.01341*, 2019.
- [12] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [13] C. Wang, J. M. Buenaposada, R. Zhu, and S. Lucey, "Learning depth from monocular videos using direct methods," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [14] R. Mahjourian, M. Wicke, and A. Angelova, "Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [15] Z. Yin and J. Shi, "Geonet: Unsupervised learning of dense depth, optical flow and camera pose," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [16] Y. Zou, Z. Luo, and J.-B. Huang, "Df-net: Unsupervised joint learning of depth and flow using cross-task consistency," in *European Conference on Computer Vision (ECCV)*, 2018.
- [17] Y. Chen, C. Schmid, and C. Sminchisescu, "Self-supervised learning with geometric constraints in monocular video: Connecting flow, depth, and camera," in *ICCV*, 2019.
- [18] R. Anurag, V. Jampani, K. Kim, D. Sun, J. Wulff, and M. J. Black, "Competitive collaboration: Joint unsupervised learning of depth, camera motion, optical flow and motion segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [19] F. Tosi, F. Aleotti, P. Zama Ramirez, M. Poggi, S. Salti, L. Di Stefano, and S. Mattoccia, "Distilled semantics for comprehensive scene understanding from videos," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
- [20] C. Godard, O. Mac 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.
- [21] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilennets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
- [22] Y. Wang, Z. Lai, G. Huang, B. H. Wang, L. Van Der Maaten, M. Campbell, and K. Q. Weinberger, "Anytime stereo image depth estimation on mobile devices," *arXiv preprint arXiv:1810.11408*, 2018.
- [23] Wofk, Diana and Ma, Fangchang and Yang, Tien-Ju and Karaman, Sertac and Sze, Vivienne, "FastDepth: Fast Monocular Depth Estimation on Embedded Systems," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.
- [24] R. Garg, N. Wadhwa, S. Ansari, and J. T. Barron, "Learning single camera depth estimation using dual-pixels," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 7628–7637.
- [25] Y. Zhang, N. Wadhwa, S. Orts-Escalona, C. Häne, S. Fanello, and R. Garg, "Du<sup>2</sup> net: Learning depth estimation from dual-cameras and dual-pixels," *arXiv preprint arXiv:2003.14299*, 2020.
- [26] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3D: Learning from RGB-D data in indoor environments," *International Conference on 3D Vision (3DV)*, 2017.
- [27] I. Vasiljevic, N. Kolkin, S. Zhang, R. Luo, H. Wang, F. Z. Dai, A. F. Daniele, M. Mostajabi, S. Basart, M. R. Walter, and G. Shakhnarovich, "DIODE: A Dense Indoor and Outdoor DEpth Dataset," *CoRR*, vol. abs/1908.00463, 2019. [Online]. Available: <http://arxiv.org/abs/1908.00463>
- [28] Z. Li, T. Dekel, F. Cole, R. Tucker, N. Snavely, C. Liu, and W. T. Freeman, "Learning the depths of moving people by watching frozen people," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4521–4530.
- [29] X. Guo, H. Li, S. Yi, J. Ren, and X. Wang, "Learning monocular depth by distilling cross-domain stereo networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 484–500.
- [30] M. Poggi, F. Aleotti, F. Tosi, and S. Mattoccia, "On the uncertainty of self-supervised monocular depth estimation," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2020, cVPR.
- [31] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.
- [32] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: <https://www.tensorflow.org/>
- [33] J. Lee, N. Chirkov, E. Ignasheva, Y. Pisarchyk, M. Shieh, F. Riccardi, R. Sarokin, A. Kulik, and M. Grundmann, "On-device neural net inference with mobile gpus," *arXiv preprint arXiv:1907.01989*, 2019.
- [34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
- [36] 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 *Design Automation and Test in Europe (DATE)*, 2019.
- [37] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 285–300.
- [38] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 573–580.
- [39] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.
- [40] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [41] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.
- [42] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, S. Kamali, M. Mal- loci, J. Pont-Tuset, A. Veit, S. Belongie, V. Gomes, A. Gupta, C. Sun, G. Chechik, D. Cai, Z. Feng, D. Narayanan, and K. Murphy, "Openimages: A public dataset for large-scale multi-label and multi-class image classification." *Dataset available from https://storage.googleapis.com/openimages/web/index.html*, 2017.
- [43] Z. 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.
- [44] C. Wang, J. Miguel Buenaposada, R. Zhu, and S. Lucey, "Learning depth from monocular videos using direct methods," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2022–2030.
- [45] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.
- [46] Z. Li and N. Snavely, "Megadepth: Learning single-view depth prediction from internet photos," in *Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [47] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [48] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, "Pixelwise view selection for unstructured multi-view stereo," in *European Conference on Computer Vision (ECCV)*, 2016.
- [49] A. Holynski and J. Kopf, "Fast depth densification for occlusion-aware augmented reality," in *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, vol. 37, no. 6. ACM, 2018.
- [50] X. Luo, J. Huang, R. Szeliski, K. Matzen, and J. Kopf, "Consistent video depth estimation," in *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, vol. 39, no. 4. ACM, 2020.