Alma Mater Studiorum - Università di Bologna

DOTTORATO DI RICERCA IN

MONITORAGGIO E GESTIONE DELLE STRUTTURE E DELL'AMBIENTE - SEHM2

Ciclo 34

Settore Concorsuale: 09/H1 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Settore Scientifico Disciplinare: ING-INF/05 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

PERCEIVING THE 3D WORLD FROM SINGLE IMAGES

Presentata da: Filippo Aleotti

Coordinatore Dottorato

Alessandro Marzani

Supervisore

Stefano Mattoccia

Co-supervisore

Luigi Di Stefano

Esame finale anno 2022

Abstract

Depth represents a crucial piece of information in many practical applications, such as obstacle avoidance and environment mapping. This information can be provided either by active sensors, such as LiDARs, or by passive devices like cameras. A popular passive device is the binocular rig, which allows triangulating the depth of the scene through two synchronized and aligned cameras. However, many devices that are already available in several infrastructures are monocular passive sensors, such as most of the surveillance cameras. The intrinsic ambiguity of the problem makes monocular depth estimation a challenging task. Nevertheless, the recent progress of deep learning strategies is paving the way towards a new class of algorithms able to handle this complexity.

This work addresses many relevant topics related to the monocular depth estimation problem. It presents networks capable of predicting accurate depth values even on embedded devices and without the need of expensive ground-truth labels at training time. Moreover, it introduces strategies to estimate the uncertainty of these models, and it shows that monocular networks can easily generate training labels for different tasks at scale. Finally, it evaluates off-the-shelf monocular depth predictors for the relevant use case of social distance monitoring, and shows how this technology allows to overcome already existing strategies limitations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We live in a world full of data. Modern technologies allow for generating and acquiring a vast amount of information rapidly. According to [107], in 2018 more than 3.7 billion humans had internet access, and used it for sending 16 million text messages, 156 million emails, posting about 47 thousand photos on Instagram or watching 4 million YouTube videos every single minute. These impressive numbers are destined to increase even more: for example, it is expected [106] that the number of photos captured in 2022 will be 1.5 trillion, against the 1.12 trillion in 2020. The availability of data, devices and sensors to acquire them and global infrastructures for sharing ignited the spread of solutions able to process them efficiently.

Nowadays, computer algorithms are a valid, and probably the only, way to handle the mass of information we produce every day. When we deal with images, computer vision helps us to mimic or even to extend human behaviour: a picture might provide hundreds of cues about the sensed scene, which can be used with success to address real problems. Environmental monitoring, traffic flow analysis and object tracking are just a few examples of practical applications in which computer vision can be employed in a non-intrusive way. For instance, suppose we have to realise a social-monitoring application, to detect and report when social distance among people is not preserved. Personal devices equipped with Bluetooth [148] or radar [128] sensors can be adopted for the task, but they require active collaboration from the users and a well-defined protocol. On the other hand, a conventional surveillance camera can provide images of the environment in real-time, allowing to realise the monitoring application without the direct involvement of the users. Another example is the inspection of structures in dangerous situations, such as underwater or precarious buildings: a robot or a drone can solve the task while preserving the safety of human operators.

Classical computer vision approaches adopted hand-crafted heuristics and strategies to face the complexity of the problems, but in the last few years learning-based solutions are leading to astonishing results. These new algorithms can learn the expected behaviour directly from the data, leveraging or not ground-truth labels for training purposes. We refer to *supervised* settings those that leverage labels at training time, while to *unsupervised* training strategies that do not require ground-truth. In between, *self-supervised* settings are those for which, at training time, we do not have labels yet exploiting additional information that will not be available at test time. An example of a self-supervised setting is employing stereo pairs to train monocular depth models, as we will see in the following. While supervised strategies achieve a better accuracy, sourcing labels is not trivial in general, and requires a large effort. For this reason, self and unsupervised approaches are particularly appealing.

One of the main limitation of pictures is that they do not preserve direct information about the 3D geometry of the sensed scene: in fact, when a real world scene is projected on the image plane, the information about the depth is lost. Nonetheless depth is crucial to solve a plethora of applications, such as augmented and mixed reality, obstacle avoidance, 3D inspection and many more. Many active sensors have been deployed in the past years, in which the device generally perturbs the nearby environment. For instance, LiDAR devices emit laser beams, Time-of-Flight (ToF) sensors retrieve the depth from the time required by the emitted signal to come back while the Microsoft Kinect projects structured light patterns on the scene. Despite their accuracy, active sensors are generally costly, suffer in scenes flooded with sunlight and might interfere with other devices deployed in the scene. To tackle these issues, a long-standing yet paramount problem in computer vision is binocular stereo depth estimation, aimed at retrieving the depth of the scene given two synchronized and rectified pairs of images. Specifically, every pixel in the left image has at most a a single corresponding pixel in the right image [1]. Moreover, the peculiar setup forces corresponding pixels to lie on the same horizontal line. The horizontal offset between matching pixels is called *disparity*, while the distance between the optical centers of the two cameras is the *baseline* b. Figure 1.1 illustrates the stereo setup. Disparity and depth share an inverse relationship: the larger the disparity, the closer to the camera the object. Since cameras are supposed to be rectified and calibrated, i.e. we know their intrinsic parameters and in particular the focal length f, we can obtain depth Z from disparity D by triangulation:

$$Z = \frac{bf}{D} \tag{1.1}$$

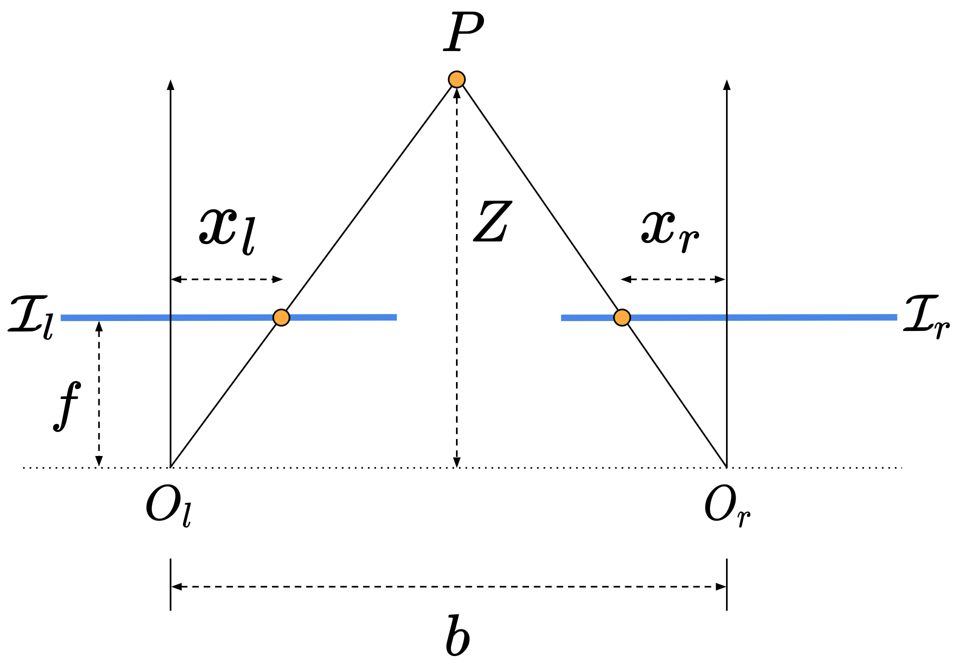


FIGURE 1.1: **Stereo setup.** The 3D point P, with coordinates (X,Y,Z), is projected on the left $\mathcal{I}_l$ and right $\mathcal{I}_r$ image planes. The distance between the two optical centers is the baseline b, while f is the focal length of the camera. The horizontal coordinates in the two planes are x_l and x_r respectively. The depth Z is inversely proportional to the disparity D, where $D = x_l - x_r$.

[1]This condition does not hold for slanted surfaces, as evinced in [213]

In practice, stereo algorithms are in charge of computing the disparity for each pixel in the image, used to obtain the depth. Once the depth Z is known, the corresponding 3D point $P = (X, Y, Z)$ can be obtained by means of *back-projection*:

$$X = \tfrac{Z}{f}(u - c_x)$$
$$Y = \tfrac{Z}{f}(v - c_y)$$

where (u, v) are coordinates of the projection of P on the image plane of the left camera and (c_x, c_y) are the camera intrinsic parameters representing the coordinates of the piercing point. The set of 3D points we obtain starting from each pixel in the image is called *point cloud*. Moreover, it is worth noticing that we are supposing that the focal length is the same both horizontally and vertically. When this condition does not occur we have to take into account the two different focals f_x and f_y when computing X and Y respectively.

The stereo problem can be generalized to the case of several views of the same scene acquired by arbitrary camera positions. We refer to this configuration as *multi-view stereo* [73], and it works under the assumption of a moving camera, both stereo or monocular, in a stationary world. Moreover, some applications are tolerant against some missing values in depth maps but largely suffer in case of errors. For instance, in the case of obstacle detection, errors in depth measures might have dramatic consequences, especially for closer objects. For this reason, confidence measures have been proposed to detect reliable estimates. Similarly, if we are interested in locating bad regions, we deal with uncertainty estimation. It is worth noticing that confidence and uncertainty are two sides of a coin, and the choice depends on the application. Confidence is generally applied as a post-processing step in classical formulations, meaning that we have to estimate disparity maps first. As we pointed out before, performing stereo means solving the problem of matching, which is a wide and fundamental problem in computer vision. Another popular matching problem is optical flow: given two images, the goal is to find the *apparent motion* of brightness patterns in the image. This problem recalls stereo matching, with the main difference that in optical flow any pixel can move in a 2D grid and not just horizontally. Under certain circumstances, the motion of pixels in the image can be traced back to the motion of objects in the world. We call this condition motion field but, even if the two are not always the same, they are often referred to collectively as optical flow [18]. In practice, optical flow has various applications in video interpolation and restoration [325], medical [330, 255], surveillance and monitoring [21, 292, 57] and many other fields. Optical flow can also be used to source valid correspondences for depth triangulation [358] from two subsequent images.

However, the information in images goes beyond geometry. When looking at an image humans recognize objects, thus can assign to each pixel a *semantic* meaning. Image segmentation, and in particular semantic segmentation, is a relevant field in computer vision and it has countless applications in medical image processing [171], precision agriculture [199] and many other fields.

Optical flow and stereo methods require at least two images. A natural question, at this point, would be: *can we infer the depth from a single image*? Nowadays, even the cheapest device has at least one camera that can acquire pictures. Thus, a monocular solution would open up depth-based applications at scale, exceeding some of the main limitations of conventional stereo (e.g. cameras for surveillance are generally monocular and fixed, so neither binocular nor multi-view stereo can be used). Unfortunately, retrieving the geometry from still images is extremely challenging since

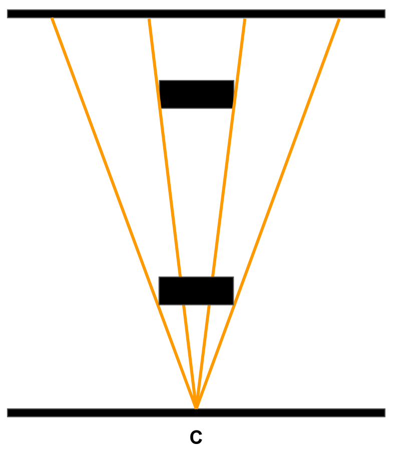


FIGURE 1.2: **Ponzo illusion.** Despite the two rectangles are equal, they appear different due to the lines that intersect at the point C.

it is *ill-posed*, meaning that, at least in theory, there exist an infinite number of possible solutions. To provide an intuition about the problem, in Figure 1.2 we depict the Ponzo illusion. Due to the different number of intersecting lines, our brain is fooled and the two identical rectangles look to be different in size.

Nonetheless, not all the infinite solutions are feasible in practice, and monocular cues can disambiguate false priors about scene geometry. Images contain light, shadows, occlusions and relative size cues that are valuable hints to tackle the issue, and it is not a coincidence that human beings can navigate the scene and avoid obstacles even with a closed eye. While coding these cues with human-based heuristics is quite challenging, learning them directly from data largely mitigate the complexity of the problem. Contextually, in the last few years Deep Learning is unlocking a new generation of depth-based applications, with particular interest for self-driving cars [113, 83], robotics [133, 202] and augmented reality [154, 88], in which a depth predictor learns to infer depth maps from single images.

This work focuses on monocular depth estimation, ranging from the design of models suited for addressing the problem to its relationship with other computer vision tasks. Specifically, in Chapters 4 and 5 we present novel networks for accurate and efficient monocular depth estimation respectively. Then, in Chapter 6 we leverage also semantic segmentation and optical flow to obtain a single model for holistic scene understanding, while in Chapter 7 we introduce a fast and low-effort pipeline to distill pseudo labels at scale, allowing to train effectively lightweight monocular models. With similar purposes, in Chapter 8 we show how off-the-shelf monocular estimators are beneficial to source optical flow labels at scale, while in Chapter 9 a peculiar monocular model is used to guide the training of a stereo model when binocular stereo pairs are available also at training time. We then tackle the problem of monocular confidence estimation: Chapter 10 illustrates different techniques aimed at estimating uncertainty of monocular predictors. Instead, in Chapter 11 we deal with stereo estimation again: in particular, the Chapter presents a neural disparity refinement model able to refine, using monocular reasoning, input disparity maps sourced by classical or even deep stereo models. Finally, the last Chapter 12 describes a practical application in which social distancing violations are assessed by means of monocular depth.

Chapter 2

Related works

After having introduced fundamental concepts, required to understand this research project, this Chapter provides an extensive overview about related works in the fields of depth estimation, with particular focus on stereo and monocular depth estimation, and optical flow. Moreover, we also review briefly relevant literature about confidence measures and semantic segmentation, and how these technologies have been adopted for monitoring applications.

2.1 Stereo depth estimation

2.1.1 Traditional approaches

Among passive strategies for depth estimation, binocular stereo is probably the most adopted, since it requires only two rectified images and it does not suffer from well-known problems of active sensors — as missing returns due to mechanical rotations, or multi-pathing. Although working only with stereo pairs induces some challenges (e.g. low texture regions and repetitive structures), nowadays this technology is mature enough, and it is quite easy to find binocular cameras even on commercial devices, such as smartphones. Scharstein and Szeliski [264] report the fundamental steps shared by many traditional stereo matching pipelines, that are:

1. matching cost computation
2. cost aggregation
3. disparity optimization/computation
4. disparity refinement

It is worth noticing that not all the steps are mandatory: *local* stereo matching algorithms, for instance, generally perform steps 1, 2, 4 while *global* ones 1, 3, 4. According to [264], local and global computation represents the most important classification for stereo algorithms: in local strategies, cost computation and aggregation are the core of the computation, and disparity computation is naively achieved through *winner-take-all* strategy; on the other hand, global approaches invest most of their computational budget in cost optimization, since they aim at finding a disparity function that minimizes the global energy. In between, *semi-global* methods share similar ideas with global strategies, but perform their computation on a subset of the points. In the following, we briefly summarise these categories.

Local methods Given the reference image $\mathcal{I}_r$ and the target image $\mathcal{I}_t$, in cost computation step we compute, for each pixel (x, y) in $\mathcal{I}_r$, the correspondence cost with all — or a subset of — the pixels $(x + d, y)$ of $\mathcal{I}_t$. In the past, different cost computation functions have been proposed — *Sum of Squared Differences* (SSD) and *Normalized*

Cross Correlation (NCC) [36, 264], just to name a few — because some functions are more robust than others in particular conditions. Similarly, transformations, such as AD-Census [351], have been proposed with the aim of improving the effectiveness of the computed cost. [298] reports a valuable evaluation of different cost functions.

After cost computation, the 3D *Disparity Space Image* (DSI) is ready, and encodes pixel similarities. At this point, we could select, for each pixel, the one with the lowest cost in the DSI, obtaining the final disparity map. Unfortunately, by doing this, the resulting map would be noisy since many pixels could have similar costs (e.g. due to repetitive patterns), and pixel-wise costs could not be representative enough (e.g. in occluded areas). To overcome this limitation, cost aggregation strategies are widely adopted, allowing to augment the cost of each pixel with those of other pixels in the DSI. This set of pixels, called *support window*, can be fixed or adaptive: in the former case, squared patches around the pixel is the default choice, while in the latter many different strategies have been proposed [215, 312], and evaluated in [299]. Finally, the winner-take-all strategy is in charge of selecting the final disparity map: for each pixel, we have to look for the minimum cost in its DSI's slice. However, the computed disparity results to be discrete since the DSI does not have a continuous formulation. Discrete maps suffer a *step-wise* effect, which could be an issue for some applications. To alleviate the problem, final refinement strategies are in charge of sup-pixel interpolation, e.g. by means of second degree functions or anisotropic diffusion [225]. If needed, this stage may also run post-processing algorithms aimed at noise-reduction [54] and hole filling [27].

Global methods Texture-less and occluded regions are hard to match, and local information may not be representative to solve ambiguities. Differently from local methods, global strategies do not perform (in general) cost aggregation but cast the matching problem into an energy minimization one. Specifically, these methods first create a graph from the image, connecting closer pixels, then they assign a label — the disparity — to each node in the graph, so that each disparity does not change excessively in the neighbourhood and it is consistent with the pixel intensities of the connected nodes.

Specifically:

$$E(D) = E_{data}(D) + \lambda E_{smooth}(D) \tag{2.1}$$

where the first term represents the sum of the matching costs of all the pixels in the image, while the second term adds a penalty for pixels with a different disparity than neighbours.

This problem, which is NP-Hard in complexity, has been tackled exploiting approximate solutions based on Dynamic Programming [23, 28], Graph Cuts [32, 31] and Belief Propagation [142], however they are not able to scale well with image size.

Semi-global methods The complexity of global methods prevents their real-time execution. To overcome this limitation, semi-global strategies have been proposed, in which Dynamic Programming or Scanline Optimization are enforced on a subset of the pixels. A popular example is Semi-Global Matching [101], in which the cost of each pixel is aggregated along S scanlines (generally 8 or 16). Specifically, the energy function along the scanline $s \in S$ is:

$$E_s(p,d) = C(p,d) - \min_i E_s(p',i) + \min \begin{cases} E_s(p',d), \\ E_s(p',d-1) + P_1, \\ E_s(p',d+1) + P_1, \\ \min_i E_s(p',i) + P_2 \end{cases} \tag{2.2}$$

where $C(p,d)$ is the matching cost, while P_1 and P_2 two penalty terms ($P_1 < P_2$). Then, the costs for each disparity value is computed as:

$$E(p,d) = \sum_s E_s(p,d) \tag{2.3}$$

Once the aggregated cost $E(p,d)$ is available, the final disparity value $\hat{d}$ for each pixel is given by:

$$\hat{d} = \underset{d}{\operatorname{argmin}} E(p,d) \tag{2.4}$$

Since each scanline is independent, SGM can efficiently exploit multi-threading, and GPU [100] and FPGA [19] implementations have been proposed.

2.1.2 Machine learning and deep learning based approaches

Classical approaches proved to be effective in tackling the stereo problem, but at the cost of highly engineered pipelines, often based on heuristics and hyper-parameters that must be tuned for each setting. For this reason, the promise made by recent machine learning and in particular deep learning strategies, who claim that is possible to learn to solve the problem directly from data, is particularly attractive, and has been largely investigated in the field. Seminal works exploited learning-based strategies to improve individual steps of the stereo pipeline [264], such as by inferring a cost function, and this paved the way towards end-to-end paradigm, which represents, nowadays, the state-of-the-art approach to the stereo problem in terms of accuracy. We are going to briefly introduce some notable methods, and further details can be found in specialized surveys as [235, 150].

Learning in the stereo pipeline. MC-CNN by Zbontar and LeCun [353] represents the most groundbreaking machine learning approach for stereo depth estimation. A Siamese network is in charge of extracting features from two input patches, then a feature similarity score can be computed with conventional feature metrics (MC-CNN-*fst*) or learned (MC-CNN-*acrt*). When costs are available, cost aggregation is performed following [101]. Park and Lee [219] learn a cost function by means of a CNN and propose a per-pixel pooling strategy which helps to add more context. Luo et al. [179] cast the problem as a multi-class classification, in which the network is able to process the full set of disparity candidates for each pixel, thus computing the features for all the right patches in a single forward pass. An inner-product layer computes the score for each disparity hypothesis, then a softmax operator turns scores into probabilities. The efficient computation of [179] requires less than a second, which is a notable improvement if compared to the 20 seconds of MC-CNN-acrt [353]. Similarly, optimization step has been improved with learning strategies: Scönberger et al. [266] propose to leverage a random forest classifier to select the best scanline for each pixel, Poggi and Mattoccia [232] reduce the streaking artifacts in SGM maps by means of a scanline confidence score while in [271] the authors exploit a CNN to predict P_1 and P_2 penalties of SGM. Finally, refinement strategies have been proposed to improve the quality of off-the-shelf stereo maps. Güney and Geiger [87] alleviate textureless and reflective areas using object-category specific disparity proposals, while in [78] the authors show that their *Detect, Replace, Refine* (DRR) framework is effective to detect errors in input maps (e.g. those from [179]). Once the errors have been detected, the framework replaces them with new labels and finally refines the map with residual corrections; Batsos and Mordohai [22] claim that recurrent architectures are more suited for disparity refinement task, and propose a residual architecture that is able to further improve its own predictions, starting from initial maps sourced by MC-CNN [353].

End-to-end architectures. Previous methods leverage machine or deep learning to improve one or more steps in the conventional stereo pipeline, but Convolutional Neural Networks (CNN) proved to be effective in learning the task in end-to-end fashion. Dispnet [191] paved the way towards this new paradigm. In their paper, Mayer et al. show that a CNN, with UNet [252] design, has enough capacity to address the stereo matching problem if trained on a large set of data. To this aim, they release the synthetic dataset SceneFlow, described in Chapter 3.1, and use it to train their model in a supervised fashion. A novel correlation layer is pivotal to obtain state-of-the-art results: taking inspiration from [61], the authors propose to aggregate the features along the horizontal scanline, providing to the network more context for each pixel. Dispnet ignited the rapid diffusion of end-to-end stereo models with astonishing results. Another milestone in the field was GC-Net, by Kendall et al. [136], who proposed a novel network able to preserve the geometry of the problem. Differently from Dispnet, in which features are collapsed while computing the cost volume, GC-Net creates a 4D cost volume, with shape $H \times W \times D \times F$, where H and W are respectively image height and width, D the number of disparity hypothesis and F the size of the features extracted for each pixel from the original image. That is, the cost volume is built by shifting and concatenating right features with left ones for each pixel. Then, the cost volume is processed by further convolutions to regularize it; at the end of the computation, the cost volume has a shape of $H \times W \times D$, and a final differentiable *soft-argmin* operator is in charge of selecting the best disparity candidate for each pixel. GC-Net proves to be more accurate than Dispnet, but it pays in higher computational costs, since the 4D DSI requires 3D convolutions instead of 2D ones, which are much more expensive. This difference creates a further taxonomy in the field: methods that follow the design of Dispnet, with 3D cost volumes processed by 2D convolutions, are called **2D networks**, while we refer to strategies that leverage a 4D cost volume as **3D networks**.

Recent 2D models focus on improving the quality of the predictions with refinement strategies [165, 217], semantic segmentation [339] or by incorporating edge cues [279, 278]. On the other hand, 3D architectures have been improved with peculiar layers, such as the *Spatial Pyramidal Pooling* proposed in PSMNet [43] or the *locally guided* and the *semi-global* aggregation layers of GANet[354], coarse-to-fine approaches [137, 317] or pruning strategies [62] for reducing the inference time or allowing high-resolution [337] processing. Particularly worthy of attention is the work from Guo et al. [90]: to alleviate computational burdensome of 3D convolutions without worsening the accuracy as in 2D convolutions, they propose the *group-wise* convolutional layer. Given the input features with shape $H \times W \times N$, the layer first splits the features into N groups along the channel dimension, then computes N correlation scores and stack them again to form a volume with shape $D \times H \times W \times N$, which is finally processed by means of 3D convolutions. Their architecture, called GWC-Net, is based on PSMNet, nonetheless it proves to be more accurate yet faster on popular benchmarks [191, 196, 76].

Previous works focus their efforts on architecture search, and follow a conventional train scheme for which an intial pre-training on large synthetic datasets [191, 303], where perfect ground-truth is available, is followed by a fine-tuning on the target dataset. This latter training, which is performed usually on real data, is required to overcome the *domain gap* issue, a well-know problem that afflicts deep stereo models. In particular, synthetic data are far from being realistic, even when realistic noise is taken into account. For this reason, networks trained on synthetic data only struggle when run on real data. Zhang et al. [355] tackle the problem with a novel normalization layer, which allows to reduce the impact of image-level domain shifts. Differently, Watson et al. [320] present a smart pipeline to train stereo models on large collections of real single images: given a single image and an off-the-shelf monocular network trained to be robust in the wild [245], they synthesize the right image of a virtual stereo pair, and use the monocular depth as ground-truth supervision to train a stereo model.

Last but not least, particularly attractive is the chance to train stereo models from raw pairs, thus without the need of training labels. In fact, the stereo setup introduces additional constraints (e.g. images are synchronized in time, epipolar lines are horizontal) that make easier to learn how to match pixels. Since no ground-truth label is available, many self-supervised methods exploit the photometric error as training signal [362, 316]. Specifically, the predicted disparity can be used to warp the target image to obtain a reconstructed reference image, and the photometric discrepancy between the real and the reconstructed references represents the training objective (that is, the better the disparity the more realistic the reconstructed image). Complementary to the photometric error, other strategies exploit left-right check [363], co-teaching [316] or traditional stereo methods [300] as additional guidance at training time, paving the way for real-time adaptation [301]. Finally, self-supervision proved to be effective also in case of multi-task models [318, 151, 126], in which stereo problem is tackled in conjunction with optical flow or semantic segmentation.

2.2 Optical flow estimation

2.2.1 Traditional approaches

The seminal work by Horn and Schunck [105] introduces the problem and proposes a variational framework to tackle it. Specifically, the *optical flow constraint* equation can be derived under the assumption of brightness consistency, but it cannot be solved uniquely since it contains more variables than linearly independent equations. To address the problem, known as *aperture*, Horn and Schunck add a smoothness term as further constraint, obtaining the final energy function E to minimize:

$$E = \iint [(I_x u + I_y v + I_t)^2 + \lambda(u_x^2 + u_y^2 + v_x^2 + v_y^2)] dxdy \tag{2.5}$$

where λ is a weight factor while u, v are the horizontal and vertical components of the optical flow vector. Instead, the subscripts x, y, t indicate derivatives along x,y and time respectively in the image domain I or for flow components u and v.

Another milestone in the field is the work from Lucas and Kanade [177], in which the authors assume that the motion is constant for all the pixels belonging to a small patch. This assumption allows to build an over-determined system of equations for each pixel, that can be approximated by means of least-squares. The proposed method works well in texture regions, where the spatial derivatives of close pixels provide meaningful information to solve the system, while it suffers if the patch contains a sharp edge or lacks of texture, because the spatial derivatives are close to zero in one or both the dimensions respectively. Both the methods ignited a prosperous literature [71, 20], and many of the original limitations have been addressed. For instance, violation of the brightness constraint has been tackled assuming as constant the gradient [35] or even higher order derivatives [218], changing the color space [198] or by applying robust transformations [93, 313], whereas pyramidal approaches [13, 195, 285] help manage large motions. However, coarse-to-fine approach could get stuck in local minimums when small objects move fast [285, 250]. [34, 336] address the problem by means of descriptor matching, in which a preliminary set of matches is further improved in a variational framework. EpicFlow [250] proposes to interpolate at the full image resolution the initial set of candidates while preserving edges, through an edge-aware term. CPM [111] improves EpicFlow introducing a coarse-to-fine matching based on random search: in each level of the pyramid, random search improves the current set of seeds looking for the best matching in a bounded range, then the seeds serve as initial flow guesses in the next level. After the last level of the pyramid, initial matches are finally optimized with EpicFlow. Conversely, RicFlow [110] overcomes the vulnerability of EpicFlow to outliers in initial matching: the scene is segmented into SLIC superpixels [4], and the flow of each region is computed using a RANSAC-like [70] algorithm, supposing that the flow of each superpixel meets a piecewise affine model; then a fast propagation mechanism between superpixels iteratively improves the models starting from superpixel initialization. Final result is obtained after a variational refinement stage, in which the global energy is minimized as in [250].

2.2.2 Deep learning based methods

Even though classical approaches generated a flourishing literature in the past, they generally require global optimizations that are hard and time consuming to solve. To tackle this issue, additional constraints (e.g. the model of the motion), heuristics and complex pipelines have been introduced. Recently, the rise of machine and deep learning solutions helped to address the problem, paving the way towards end-to-end models able to learn directly from data. [254] [286] [293]

Flownet [61] represents the first, notable attempt towards end-to-end models for optical flow. Inspired by the recent progress in image classification, depth estimation and semantic segmentation fields, the authors propose a CNN able to predict optical flow labels starting from two images. Their model, called FlownetS, contains only convolutional layers and are optimized by means of gradient descent. Moreover, they also propose a more accurate model named FlownetC, which exploits a correlation layer to perform matching in the feature space: given two feature maps $\mathbf{f}_1$ and $\mathbf{f}_2$ extracted from the images, the correlation layer computes the scalar product between two patches centered in x_1 in x_2 respectively in $\mathbf{f}_1$ and $\mathbf{f}_2$

$$c(x_1, x_2) = \sum_{\mathbf{o} \in [-k,k] \times [-k,k]} \langle \mathbf{f}_1(x_1 + \mathbf{o}), \mathbf{f}_2(x_2 + \mathbf{o}) \rangle \tag{2.6}$$

The correlation scores are then stacked with other features extracted from the first image and then used in the decoding stage. Specifically, a decoder based on upconvolutions in charge of predicting the final optical flow map at the original resolution. To train their model, the authors also generate the synthetic dataset Flying Chairs using real texture from Flickr and 3D assets of chairs. This dataset is presented in

Chapter 3.1. FlowNet obtained impressive results both in terms of accuracy and execution time (since convolutions can easily exploit GPU for computation), but it suffers in case of small displacements. To this aim, Ilg et al. [120] first studied how to schedule multiple datasets at training time, then they proposed FlowNet2, a novel architecture made of multiple FlowNetC and FlowNetS. Two main branches are in charge of computing large and small displacements respectively. On the one hand, computing large motions requires more capacity, thus more FlowNet networks are connected in series. In each stage the second image is warped according to the previously predicted optical flow, providing to the network an additional feedback about the goodness of the flow. On the other hand, small motion requires a more fine-grain analysis. Therefore the authors adopted a peculiar network trained on a synthetic dataset specifically designed for small displacements. The two branches are fused together in the last stage, producing flow maps with sharp motion boundaries and able to preserve fine details.

The well-known pyramidal approach has been used also in combination with deep-learning solutions to address large motion displacements while reducing the size of the models. Ranjan and Black [246] propose SpyNet, a coarse-to-fine network 96% smaller than FlowNet, in which each level of the pyramid learns a residual flow from the upsampled flow from the previous level and the two images (I_1, I_2), with I_2 warped using the optical flow as in classical formulations [285]. PWCNet by Sun et al. [287] further improves the basic concepts below SpyNet: besides coarse-to-fine design and warping, PWCNet introduces the cost volume as fundamental data structure to store matching costs. Differently from SpyNet, PWCNet warps features $\mathbf{f}_2$ according to the past flow, and use them with $\mathbf{f}_1$ to build the cost volume of the current pyramid level using a feature correlation layer. Moreover, while in FlowNet the cost volume is computed once so it has to employ a large search window, the pyramidal design of PWCNet allows to employ small windows in each level, thus reducing the complexity of the model. The cost volume is finally processed, together with I_1 and the upsampled flow from the previous level, to predict the optical flow of the current level. When the last level of the pyramid has predicted its flow, a final refinement network tries to ameliorate it, resembling post-processing stage adopted in traditional methods. PWCNet is 17 times smaller and 2 times faster than FlowNet2, but provides more accurate maps on conventional benchmarks [196, 37]. Similar strategies have been employed also in LiteFlowNet [117, 116, 115], paving the way towards fast methods for optical flow.

Recent state-of-the-art strategies exploit iterative schemes instead of the coarse-to-fine approach. Hur and Roth [118] replace the multi-scale decoder of PWCNet and FlowNet with a single decoder that is used at each level to estimate a residual correction for the flow. RAFT [294], on the contrary, mimics the steps of an iterative optimization algorithm: GRU units are in charge of refining an initial optical flow map for a large number of iterations. In general, the initial map is set to zero, but this is not mandatory: for instance, when processing video sequences, the *warm-start* provides a stronger initialization since the optical flow computed for the previous pair of frames is forward projected to the current one and used as initial map. Moreover, RAFT does not compute correlation scores as in FlowNet, but it takes the dot product between all pairs in $\mathbf{f}_1$ and $\mathbf{f}_2$, generating a 4D cost volume with shape $H \times W \times H \times W$. The cost volume is aggregated by means of pooling layers, generating a multi-scale structure that contains information for both small and large displacements. A *look-up* operation allows to extract information from each cost volume given the current flow estimate, then these correlation scores are stacked together and used by the iterative layer to predict a better a residual correction for the flow map.

RAFT provides state-of-the-art results, especially when it is trained in a supervised fashion. Nevertheless obtaining ground-truth labels for optical flow is extremely expensive, since no sensor is able to provide these labels directly from images. Depth sensors can be used to project the 3D scene into the image plane (obtaining the *motion field*), however expensive post-processing interventions are still required to obtain ground-truth values. For instance, the pipeline adopted in the KITTI 2015 dataset [196] to obtain sparse optical flow annotations for the 200 image pairs requires CAD models for the dynamic objects (e.g. cars) and also some manually annotated correspondences between the CAD model and the object in the image. It is easy to notice that these constraints are impractical for datasets with thousands or millions of images. To tackle this issue, in the last few years the research community puts much efforts into solutions that do not need expensive labels at training time. UnFlow by Meister et al. [194] was a first step towards this direction: their model, inspired by FlowNet and FlowNet2, is trained using an unsupervised loss which takes into account a brightness, a smoothness and consistency check terms computed using both the forward and the backward flows (i.e. the flow from I_2 to I_1). Specifically, the brightness term imposes that two corresponding pixels, if not occluded, should be similar in the image. To compensate for illumination changes, the census transform is applied on the images. The second term imposes a second order smoothness on the flow, while the third requires that, for non-occluded pixels, the forward and the backward flows should be consistent for corresponding points. Instead of masking out occluded pixels, Liu et al. [169] propose a teacher-student approach aimed at providing a valid supervision on such difficult regions: first the teacher model is trained without ground-truth supervision on non-occluded pixels, then a student network is trained employing as additional labels the outcomes of the teacher model. Specifically, the teacher first infers the optical flow from I_1 to I_2, then the two images are cropped and given as input to the student. Doing so, some pixels result to be occluded in the cropped patch, but could be visible in the original image. When this happens, the predictions from the teacher are a reliable source of supervision to train the student model to be robust against occlusions. SelFlow [170] further explores teacher-student paradigm: given a dataset without labels, a teacher model is trained in non-occluded regions exploiting a photometric loss, then the images of the dataset are perturbed by noise, which is used to hallucinate occlusions, and given as input to the student model. Similarly to [169], the student relies teacher's predictions in virtually occluded regions, making it more robust. Moreover, the authors also propose a model, based on PWCNet, that takes three consecutive images as input. The additional image could provide beneficial information about pixels that might be occluded in the future frame but not in the past, allowing the multi-frame model to build a cost volume based on forward and backward flows in each stage of the pyramid. The multi-frame approach for optical flow proved to be beneficial both for unsupervised [125] and supervised [249] strategies. Recently, Jonschkowski et al. [130] analyze key components for unsupervised optical flow. Among them, the census loss proves to be a valid objective for this task, forward-backward consistency helps to address occluded regions and second-order smoothness term provides a good regularization. These findings have been coupled with other strategies (e.g. self-teaching scheme for self-supervision and cost volume normalization) achieving impressive results. Stone et al. [283] introduce a multi-frame self-supervision to train RAFT without the need of ground-truth labels.

2.3 Confidence estimation

As evinced before, many applications require depth labels in their pipeline to solve higher level tasks, such as obstacle avoidance. Nonetheless, a valuable question that may arise is *can we trust on these predictions?* In fact, especially for critical applications, errors in the system might be fatal. For this reason, confidence estimation algorithms have gained popularity in the field, providing valuable methods to detect and remove points in depth maps that are likely to be outliers. Following the taxonomy proposed in [231], stereo confidence measures can be classified as *hand-crafted* or *learned*. In general, hand-crafted measures take into account the cost volume [190, 265], a part of it [138, 238] or even the disparity map [232, 92] as main source of information. In addition, consistency checks, as the Left-Right Consistency [63], other constraints, as the Uniqueness [59], or considerations about image properties [220, 221] might help to detect errors and challenging regions. On the other hand, learned strategies leverage classifiers to infer a confidence score for each pixel. Initial attempts employed Random Forests [33] fed with the outcomes of hand-crafted measures [92, 236], but recent solutions involve CNNs to process the disparity [304, 233] or the cost volume [193, 86]. Hu and Mordohai [109] evaluate 17 confidence measures using popular benchmarks as Middlebury [263], while Poggi et al. [238] further improve the evaluation by considering more than 52 algorithms. Similar studies have been conducted even for optical flow [145, 185].

2.4 Semantic segmentation

As evinced so far, depth information is crucial to solve geometrical problems, e.g. augmented reality or obstacle avoidance. Nonetheless, in many applications we also need *semantic* cues about the scene: for instance, we might be interested in recognizing pixels in the image that belong to the road, trees and people because we want to apply different heuristics depending on the class. In this case (and many others), we have to deal with the semantic segmentation task, since we need to understand the content of the image at pixel level. Specifically, given a set of classes (e.g. *car*, *tree* and *road*), we have to assign the best class to each pixel in the image. Seminal strategies adopted machine learning approaches, such as Random Forest [270] or Conditional Random Fields (CRF) [228], but the real turning point for this task was the introduction of deep architectures. Fully-convolutional networks [174] are the most embraced paradigm, in which a backbone first extracts features at different resolutions, then a decoder with deconvolutions predicts the semantic label for each pixel at the original resolution. Badrinarayanan et al. [17] adopt a multilevel decoder, while Zhao et al. [357] propose a pyramid pooling module to add global context. In this field, DeepLab family gave a large contribution. In the initial proposal [50], atrous convolutions help to increase the *field-of-view* efficiently, and a final CRF post-processes the predictions of the network. Then, atrous spatial pyramidal pooling [49, 47] and a new model found by neural architecture search [167] largely improve the accuracy, dismissing the need for CRF post-processing. More details about other notable works in the field can be found in the survey by Minaee et al. [200]. These methods require large datasets with accurate labels for training purposes, and a great effort has been made by the community in the past years. For instance, Microsoft COCO [166] contains 2.5 million labeled instances for a total of 328,000 images, while OpenImages [147] provides, in its v6 version, about 9 million images with rich annotations. However, none of these datasets can infuse the global knowledge a model might need at scale. To tackle this problem, *domain adaptation* strategies have been proposed, allowing to reduce the shift between training and testing domains. When labels for the target domain are not required, adaptation is called *unsupervised* [297], and represents an intriguing research direction. Besides semantic segmentation, in some application we have to identify unique instances in the images. For example, two pixels might have the same semantic meaning (e.g. person) yet belong to two different instances (two different persons). In this case, we need an instance segmentation algorithm [97, 227, 164]. Furthermore, the two tasks of instance and semantic segmentation can be combined, providing to each pixel the semantic and the instance classes. In this case, the segmentation is called panoptic [141], and different methods have been proposed [140, 53, 333], paying attention also to efficient computation [323, 104].

2.5 Single image monocular depth estimation

While stereo and multi-view problems have been largely addressed in the past years, monocular depth estimation, i.e. estimating the depth of the scene given a single image of it, is a recent challenge. In fact, at least in theory, an infinite number of 3D scenes might generate the captured image. To deal with the complexity of the problem, preliminary attempts exploited Markov Random Fields [261], Boosted Decision Trees [102] or KNN queries on a labeled database [134], but it was the introduction of CNN that ignited the research on the topic. Since CNNs require large amount of training data, with or without labels. In the following, we briefly revisit notable proposals in both supervised and self-supervised setups.

2.5.1 Learning with depth supervision

In their seminal work, Eigen et al. [65] use a CNN to predict in end-to-end fashion the monocular depth. Specifically, a global *coarse-scale* network is in charge of regressing a coarse depth map for the scene, while a subsequent *fine-scale* network takes the coarse map as input and improves it locally. The two networks are trained in supervised manner with a scale invariant loss aimed at reducing the issue given by the global scale. Indeed, the authors noticed that a non-negligible portion of the total error depends on how well the mean depth is predicted. Conversely, the scale-invariant loss allows to measure the relationship among corresponding pixels up to a scale factor, since it removes the global scale in depth maps. Following works largely improved the accuracy: Fu et al. [72] recast the conventional regression task as an ordinal regression one by means of a spacing-increasing discretization, Yin et al. [345] enforce geometric constraints including a loss on virtual normals while Bhat et al. [24] generalize the ordinal regression introducing bins that change depending on the scene. Fast computation has been addressed by Wofk et al. [326]: their model called FastDepth employs a lightweight encoder-decoder structure, and a final pruning further reduces the computational complexity. The previous methods employ ground-truth depth annotations at training time, but as in the stereo case these labels are expensive to collect at scale. For this reason, many works tried to overcome this limitation, proposing less expensive yet effective ways to obtain pseudo-labels (i.e. labels that are not the output of a depth sensor). Li and Snavely [163] process internet photo collections obtaining a raw depth supervision by means of structure-from-motion [267] and multi-view-stereo [268] algorithms. Since these depth maps might contain artefacts due to moving objects, or errors near depth discontinuities, a further refinement strategy is applied. This strategy takes into account the semantic segmentation of the image, used to categorize pixels into *foreground* (e.g. people, statues) and *background* (e.g. buildings, road) classes, and exploits additional constraints to ensure more consistent depth labels. Even if moving objects represent a serious issue for multi-view approaches, since they break the *moving camera in a static world* assumption, it is highly likely that real sequences collected in the wild contain them. The problem is still open, but it has been addressed successfully in constrained setups. For instance, Li and Snavely [161] leverage online videos of *frozen* people to train a monocular model able to generalize on conventional videos with moving people. Specifically, at training time the authors processed with COLMAP [268] sequences of people while doing the *mannequin challenge*, i.e. people are motionless while the camera moves in the environment. This is a favourable condition for COLMAP, which provides reliable pseudo ground-truth depth labels for both people and background objects. At test time, the model trained on pseudo labels receives real videos with moving people as input. Since the network processes the video one frame after the other, people appear as static in each frame, so the model can generalize well. Ranftl et al. [245] face the problem of training with different sources: in fact, heterogeneous datasets are crucial to obtain robust models, but at the same time the supervision they provide might be quite different. The authors mixed 3D movies, online stereo videos and large structure-from-motion collections, obtaining a large dataset depicting various conditions. A novel robust loss and a training schedule have been proposed to handle this huge diversity, allowing to train a large convolutional model with impressive generalization capability. In a following work, Ranftl et al. [244] include more training data and replaced the convolutional backbone with a vision transformer [60]. Their Dense Prediction Transformer (DPT) achieves outstanding results in monocular depth estimation and semantic segmentation tasks. Xian et al. [331] employ a pair-wise ranking loss to train on large-scale datasets, enforcing random, edge and instance guided sampling strategies. Monocular models can provide us the depth given a single image, but to obtain the 3D point cloud we also need camera parameters, which are likely to be unknown when using images sampled from the web. Yin et al. [346] address the problem and propose a model able to learn from large-scale datasets but also to infer a guess about the focal length of the camera, obtaining at the end a more realistic point cloud.

2.5.2 Learning from raw images

It is easy to imagine that collecting and preparing a large amount of data for training purposes are time-consuming tasks. On the contrary, exploiting raw images allows for notably decreasing these payloads, at the cost of weaker supervision. Self-supervised strategies use as main signal at training time the *photometric* supervision given by one or more images not available for test: with the predicted depth, a new image is built leveraging image warping process, then the result is compared with a real image captured by the camera. The discrepancy between the two images indirectly measures the quality of the depth, since a reliable depth map could synthesize a realistic image. Of course, this supervision is weaker than using depth sensors, since even a wrong depth prediction could generate a low-error pixel intensity in ambiguous regions (e.g. textureless areas). The image used for warping can be the right sample in a stereo pair or another frame in a monocular video. Notice that we need this additional source of information because a single image contains cues that are extremely difficult to exploit without supervision (e.g. defocus or occlusions), so unsupervised schemes are uncommon. Thankfully, the additional information required by self-supervised methods is cheap and easy to obtain most of the times.

We now describe notable strategies for both the families, even if proposals that combine both the strategies exist.

Stereo pairs. Garg [75] et al. advance a framework for self-supervised training of monocular models leveraging stereo pairs at training time. Specifically, the depth for the left image is turned into a disparity, using known camera parameters, and used to warp the right image. Doing so, we obtain a reconstructed left image valid for loss computation. Godard et al. [79] improve the image formation model in several ways, proposing the MonoDepth framework. First, the authors employ a bilinear sampler [122] in warping stage, replacing the Taylor expansion introduced by [75] and thus obtaining a loss that is easier to optimize. Moreover, instead of the $\mathcal{L}_2$ loss adopted in [75], in MonoDepth the loss takes into account also the *structural similarity index measure* (SSIM) [319] and a left-right consistency regularization, improving the quality of predictions. Finally, a post-processing step leads to more pleasant maps with lower errors. MonoDepth represents a milestone in the field, and it has been used as baseline for further improvements. Among them, semantic reasoning [352], virtual trinocular assumption [237], classical stereo guidance [321] generative adversarial approach [8, 327] or real-time performance [230].

Monocular videos. Training models with stereo pairs requires a stereo device, which is less expensive than manual labeling but still represents an additional constraint. To avoid this, Zhou et al. [365] leverage monocular videos at training time, thus adopting the same device both for training and testing. The depth model is trained together with a pose network, in charge of predicting the camera extrinsic parameters between two consecutive frames. However, dynamic objects break the non-rigid scene motion assumption, and for this reason the authors also employed a motion explanation mask to remove problematic regions at training time. Godard et al. [80] boost the paradigm proposing MonoDepth2, a novel framework able to use monocular videos and eventually also stereo pairs for training purposes. A better loss, which takes into account the minimum error term among multiple sourcing images, some architectural improvements and a non-learned mask allow MonoDepth2 to achieve impressive results. Consecutive works aim at incorporating optical flow [347, 52, 178], semantic segmentation [85, 41], geometric constraints [187, 257] or uncertainty estimation [340] to obtain even better results. Many efforts have been spent in improving the capacity of the networks. Johnston and Carneiro [129] add the attention mechanism in the MonoDepth2 framework, Guizilini et al. [84] introduce 3D packing and unpacking blocks aimed at preserving and recovering spatial information while Ramamonjisoa et al. [242] enforce model efficiency by means of wavelet decomposition. Further improvements are focused in excising the knowledge about camera parameters [82, 309] or pose [358], dealing with indoor environments [349, 157] or dynamic objects [159].

2.5.3 Video at test time and adaptation

In many applications we are allowed to collect video sequences at test time. Despite this, previous methods process the video as a collection of single frames, so they are not able to fully exploit internal connections among video frames. On the other hand, a model that takes multiple frames at test time can exploit the larger context to predict more reliable maps than single-image strategies. Watson et al. [322] propose a monocular model that successfully exploits video sequences also at test time. Their framework, called ManyDepth, takes more frames as input and creates a cost volume, using a learned pose for warping. Moreover, the authors also leverage a monocular model at training time as teacher model to tackle dynamic objects. Finally, also classical methods for simultaneous localization and mapping (SLAM) can be adopted to obtain expert models: Tiwari et al. [296] propose a self-improving loop in which a monocular depth model [80] and a SLAM technique [205] collaborate together to mitigate the their limitations.

In general, the weights of the model are *frozen* at test time, i.e. the model obtained at the end of the training is deployed as a set of constant parameters, but the availability of sequences allows to use self-supervised strategies suited for monocular video learning also at test time. This test time refinement was originally proposed in [40], and further investigated in several other works. Chen et al. [52] propose two different online optimization strategies: in *parameter fine-tuning* the model is optimized computing a test-time loss, while in *output fine-tuning* we directly minimize the output without recomputing the network. McCraith et al. [192] evaluate which modules receive more benefits from adaptation, studying the impact of different choices (e.g. learning rate, optimization steps) both in instance-wise and sequential adaptation. In the former case, the weights are modified individually for each image, while in the latter the model is updated throughout the whole sequence. To prevent catastrophic forgetting, replay-buffer strategy [149] and online statistics adapters [356] have been applied. Optical flow and more complex adaptation strategies for online optimization can be adopted, as proved by [160]. Finally, another important direction of research consists in obtaining an expert model for a given video sequence. These methods release fast-adaptation and frame-order constraints, enforcing long training on the given video sampling frames with arbitrary order. The goal is to obtain a model that perfectly explains the current video, generating consistent depths among all the frames. For this purpose, Luo et al. [180] propose Consistent Video Depth (CVD) framework, in which the depth of corresponding points is forced to be consistent. An off-the-shelf optical flow network is in charge of predicting pixel matching between frames, but since a video might contain several frames, computing optical flow for every pair of frame is not feasible. For this reason, the authors adopt a hierarchical frame sampling strategy, largely decreasing this cost. Kopf et al. [146] subsequently face one of the main limitations of CVD, that is the motion of dynamic objects. Their robust version of CVD joint estimates depth deformation and camera poses from the input video, while a geometry-aware depth filter recovers fine-details.

2.6 Monitoring applications and deep learning

Among the countless fields that have benefited from the recent improvement and spread of digital technologies, monitoring represents a crucial topic. Structural health monitoring, for instance, aims to detect in advance critical damages to buildings and infrastructures. It enables to assess their level of safety and integrity [114], and to operate early avoiding potential disasters. Similarly, environmental monitoring gathers all the information that reveals the actual state of a particular ecosystem. Water quality [1] and wastewater [95], air pollution [132] and soil erosion [30] are just few examples of environmental monitoring applications. Of course, practical cases for monitoring are innumerable and can involve dynamic situations occurred in a long time period, as for wild animals monitoring in their environment [306].

Modern monitoring applications exploit heterogeneous sensors connected through the internet to obtain a large amount of data, often collected with real-time constraints. For this reason, monitoring tasks can be seen as concrete applications of the *Internet of Things* (IoT) concept, and are becoming pervasive [3, 282, 2, 152, 188]. Among the sensors, cameras are widely adopted because they are cheap, can be easily deployed in many environments [292] and are supported even by low-power devices [173]. When the area to cover becomes extremely large, as for forest variable estimation [289], aerial images are frequently employed [214, 315]. The high-resolution of aerial images, however, may be critical for memory-hungry strategies, and efficient methods have been proposed [99]. Similarly, researchers and engineers applied methods based on computer vision also for other challenging situations, such as underwater [260] or in snow-covered environments [68].

The recent spread of deep learning strategies has successfully involved also monitoring [366, 203, 350], and vision-based systems [344] are achieving astonishing results for crack detection [259, 210], defect localization [224] and risk assessment [359]. Moreover, also depth [67, 123], optical flow [328, 91, 292] and semantic segmentation [44, 172] tasks have been employed successfully for improving monitoring applications. A notable example is social-distance monitoring, a crucial task due to the COVID-19 pandemic. As we will discuss in more detail in Chapter 12, semantic segmentation and depth are two vital pieces of information for addressing some intrinsic limitations of the problem. For instance, Aghaei et al.[5] require that the principal plane in the scene, such as the floor, is visible and detectable, but this condition is hard to fulfil in practice. Similarly, also wearable devices like Bluetooth bracelets [148, 144] or smartphones [128] are difficult to apply at scale, because they require an active involvement of the users. On the contrary, a passive monitoring application based on already deployed surveillance cameras represents a low-cost yet broadly usable alternative to address the problem.

Chapter 3

Datasets, metrics and baseline models

Before diving into the core of this work, first we describe the datasets and the metrics used in the field. Then, we recap some of the most important models for monocular depth estimation.

3.1 Datasets

We now introduce some of the mostly adopted datasets in the field. Disparity maps are colored using the jet colormap, so the larger the disparity value the warmer the color. Optical flow, instead, is encoded with the color wheel defined in [61] and depicted in Figure 3.1, in which the hue and its saturation are ruled by the angle and the magnitude of the flow vector respectively.

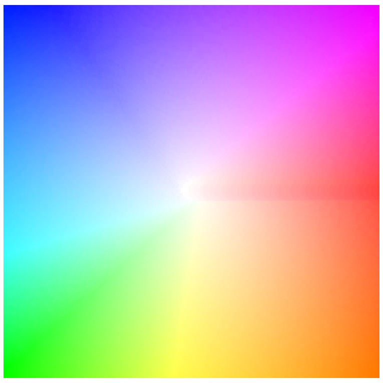


FIGURE 3.1: **Optical flow color wheel.** Each pixel is encoded with a particular color depending on its optical flow value. Specifically, the hue and the saturation depend respectively on the angle and the magnitude of the flow vector.

SceneFlow. The SceneFlow dataset [191] is a popular synthetic dataset containing around $35k$ stereo pairs with dense ground-truth maps for disparity, optical flow and disparity change tasks. The resolution is 960×540. It is composed of three different splits, called Driving, FlyingThings3D and Monkaa respectively. Figure 3.2 presents a qualitative example from FlyingThings3D for both disparity and optical flow tasks.

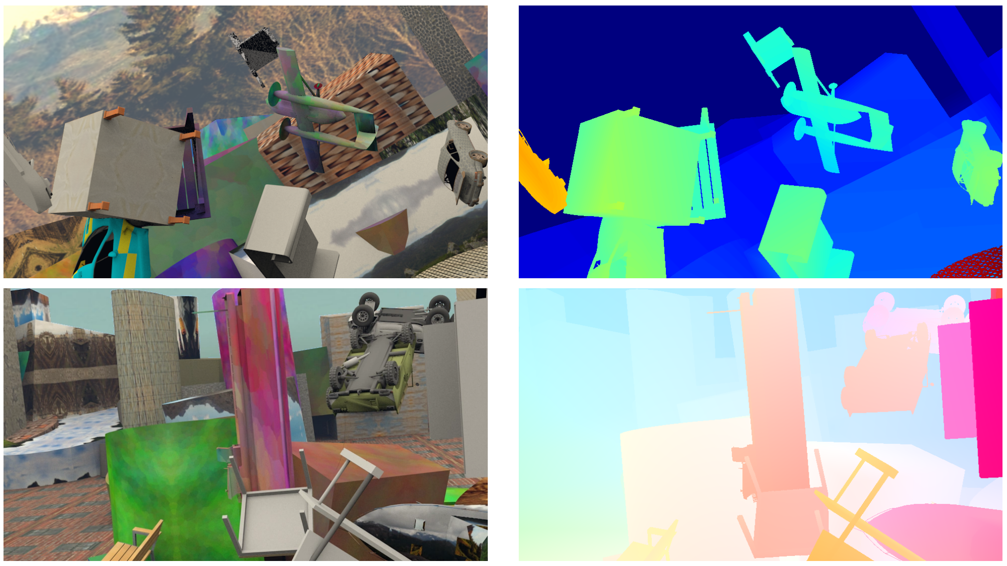


FIGURE 3.2: **Qualitative example from SceneFlow.** From left to right, the reference image and the ground-truth map for two samples from the FlyingThings3D split. First row depicts the left image of the stereo pair and the ground-truth disparity map, while second row illustrates the reference image and the optical flow map.

KITTI. The KITTI dataset is a large collection of stereo pairs suited for autonomous driving applications. In particular, it frames driving scenarios, and it has been collected by a moving car with a LiDAR device mounted on top. Due to the configuration of the system some images are slightly different in size, but the most common is 1242 × 375. The dataset divides into KITTI 2012 [77] and KITTI 2015 [196] versions. Figures 3.3 and 3.4 depict respectively samples from KITTI 2012 and KITTI 2015 respectively.

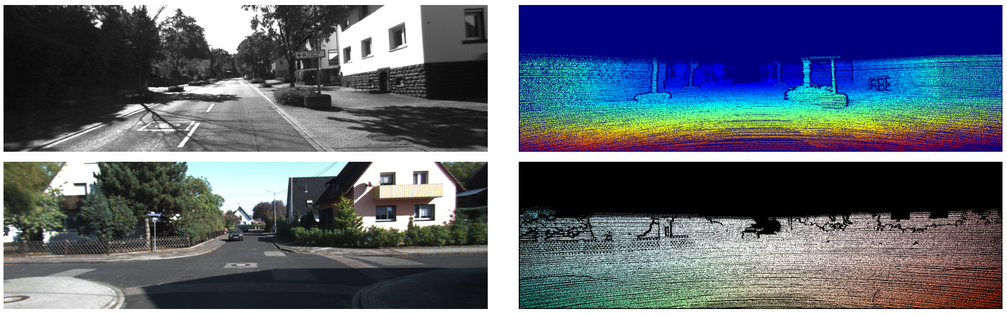


FIGURE 3.3: **Qualitative example from KITTI 2012.** From left to right, the reference image and the ground-truth map. First row depicts the left image and the disparity map, while second row presents the first image and the optical flow map.

KITTI 2012 contains 194 and 195 samples respectively for training and testing, while KITTI 2015 has 200 pairs with accurate ground-truth labels for training and 200 for testing. In particular, the KITTI 2015 split provides labels for many tasks, such as optical flow, disparity estimation, semantic segmentation, visual odometry and more, and offers an online benchmark (for which ground-truth labels are not available) to test new methods and compare them with previous strategies in the literature. Aside from labeled pairs, the dataset offers a large unlabeled split [76] made of 151 scenes with raw LiDAR measurements (i.e. the outcome of the sensor without any particular filter to remove outliers or to make the maps denser) and

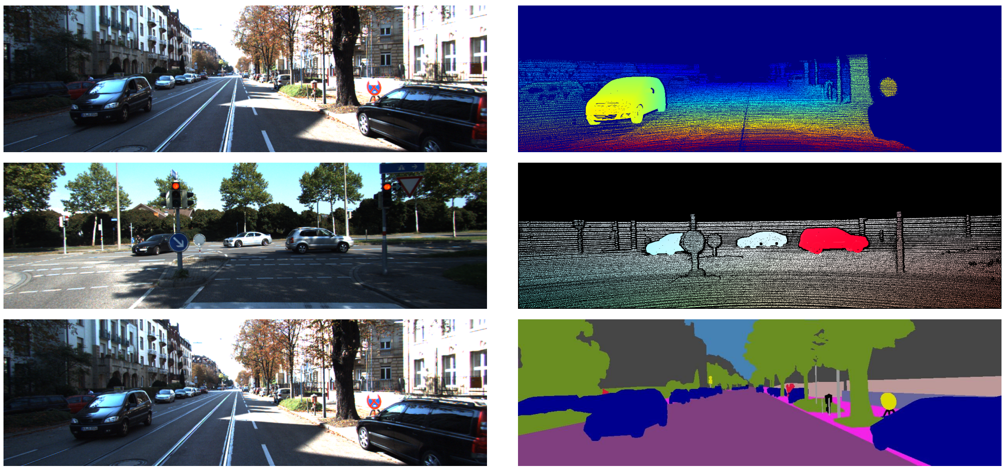


FIGURE 3.4: **Qualitative example from KITTI 2015.**From left to right, the reference image and the ground-truth map. First row depicts the left image and the disparity map, second row presents the optical flow map while third row illustrates the semantic segmentation map.

RGB stereo pairs. This split is suitable for training unsupervised strategies, due to the large number of samples. Another widely adopted way to split the KITTI dataset has been proposed in the seminal work of Eigen et al. [65]. This split, called Eigen split, uses 29 scenes of the dataset to sample 697 images for testing purposes, while the remaining 32 scenes, counting about 23,488 frames, are used for training and validation. The Eigen split is particularly adopted by the monocular depth estimation community, since it provides 22,600 images used to train monocular models in self-supervised fashion.

DrivingStereo. The DrivingStereo dataset [338] comprises 180,000 stereo pairs at 1762 $\times$ 800 depicting driving scenarios under different weather conditions. Similarly to KITTI, the car used to collect the data is equipped with a LiDAR sensor, allowing to obtain accurate depth measurements but, to obtain dense yet accurate ground-truth depth maps, the authors relied on a neural network to aggregate multiple LiDAR point clouds into a single map. Moreover, the dataset does not offer labels for other tasks, making it appropriate for training unsupervised and self-supervised methods. A qualitative example is shown in Figure 3.5.

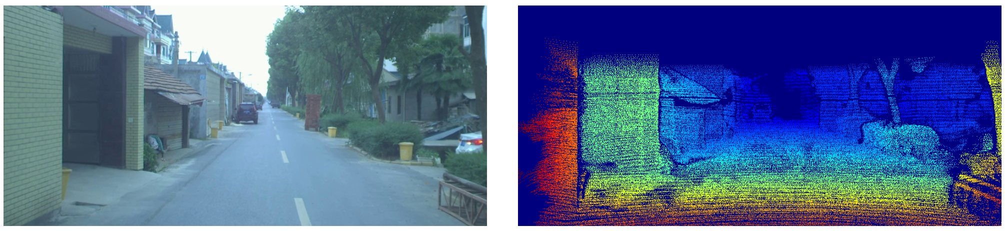


FIGURE 3.5: **Qualitative example from DrivingStereo.** From left to right, the left image of the stereo pair and the ground-truth disparity map.

Cityscapes. The Cityscapes dataset [55] depicts 50 German cities under different weather conditions, counting 25,000 stereo pairs framing driving scenes with an image resolution of 2048 $\times$ 1024. The dataset is used mostly for the semantic segmentation task, since it provides coarse semantic labels for 20,000 images and accurate semantic labels for the remaining split. Differently from KITTI or DrivingStereo, the acquisition system was not equipped with a LiDAR sensor, and the depth labels provided by the authors are the outcomes of a passive stereo method [101]. For this reason, this dataset is suited for training, aside from semantic segmentation networks, self-supervised depth or optical flow methods. Figure 3.6 illustrates the disparity map and the corresponding semantic labels for a sample of the dataset.

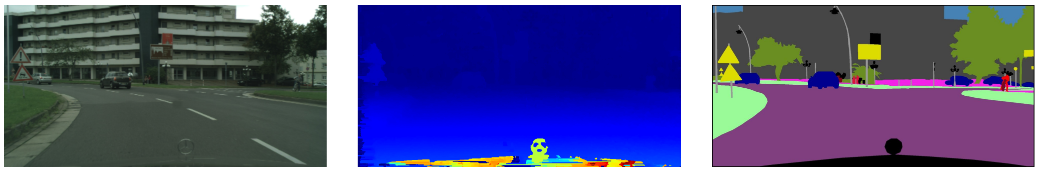


FIGURE 3.6: **Qualitative example from Cityscapes.** From left to right, the reference image of the stereo pair the disparity map and the ground-truth semantic segmentation map provided by the authors.

Middlebury v3. The Middlebury Stereo Dataset v3 [263] is a widely adopted high-resolution indoor real-world stereo dataset. It provides accurate ground-truth disparity maps at different resolutions: full-res (F), half-res (H) and quarter-res (Q), with full-res maps are roughly 5 Mpx (i.e. image size is $\sim 2900 \times 2000$). The limited training set, made of 15 images, makes the dataset suitable for generalization purposes, i.e. it allows checking if a pre-trained model can handle images taken from different domains. We depict a sample of the dataset in Figure 3.7.

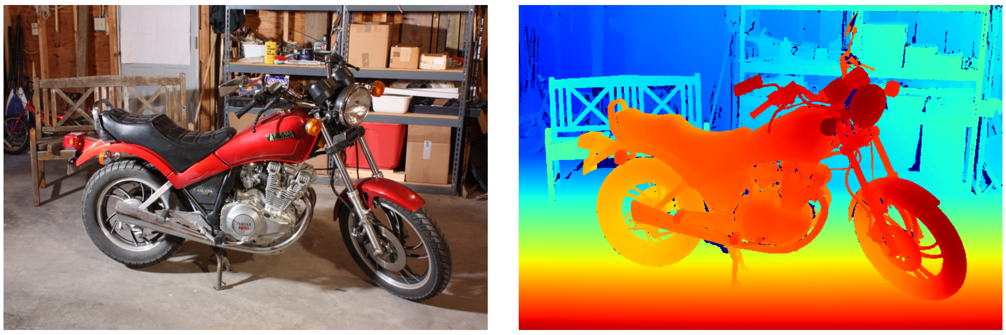


FIGURE 3.7: **Qualitative example from Middlebury.** From left to right, the left image of the stereo pair and the ground-truth disparity map.

ETH3D. The ETH3D dataset [269] depicts both indoor and outdoor environments and is made of 27 grayscale stereo pairs with ground-truth disparity maps. As for Middlebury v3 dataset, the size of this collection makes it appropriate for generalization purposes rather than for training. Images are at low resolution and can have different size, such as 713×438 or 940×491. Figure 3.8 shows a sample from the dataset.

TUM. The TUM RGBD (3D Object Reconstruction category) dataset [284] contains indoor sequences captured using a camera and a Microsoft Kinect device. Image resolution is 640×480. This dataset is a popular benchmark for SLAM algorithms, but it can be also used to evaluate generalization capabilities of monocular depth models. For instance, Li et al. [161] leverage 1815 images of this dataset for evaluation purposes. Figure 3.9 depicts a sample of the dataset.

NYU v2. The NYUv2 dataset [206] is an indoor dataset with depth measures acquired by a Microsoft Kinect device. It provides more than 400k raw depth frames and 1449 densely labelled frames at 640×480. The official test split containing 654

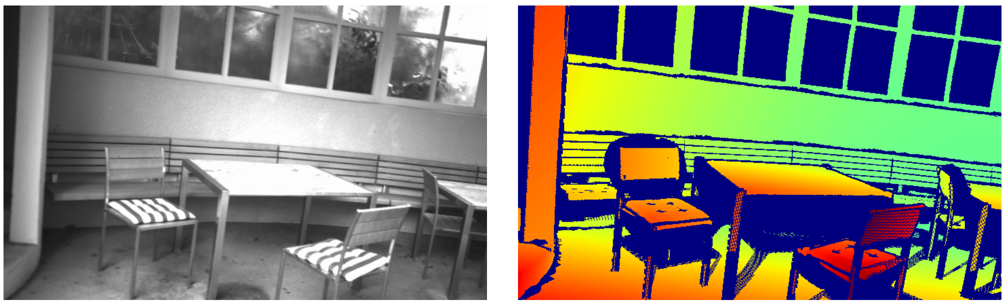


FIGURE 3.8: **Qualitative example from ETH3D.** From left to right, the left image of the stereo pair and the ground-truth depth map, colored as inverse depth.

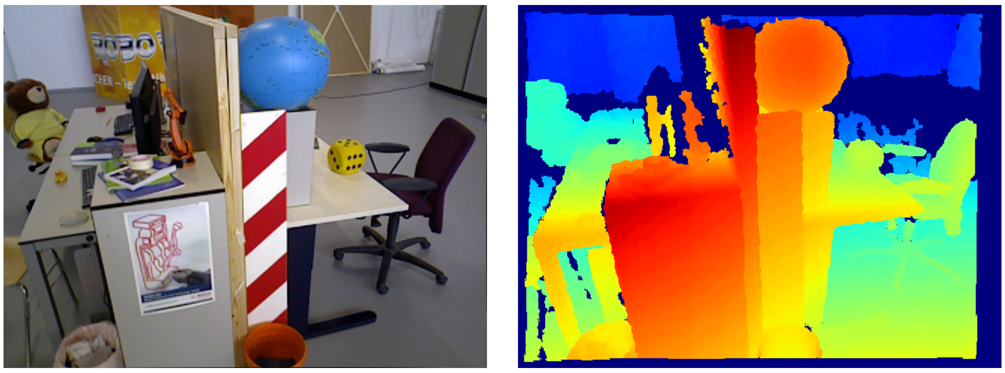


FIGURE 3.9: **Qualitative example from TUM.** From left to right, the reference image and the ground-truth depth map, visualized as inverse depth.

images, used for generalization purposes. Figure 3.10 presents a sample from the dataset.

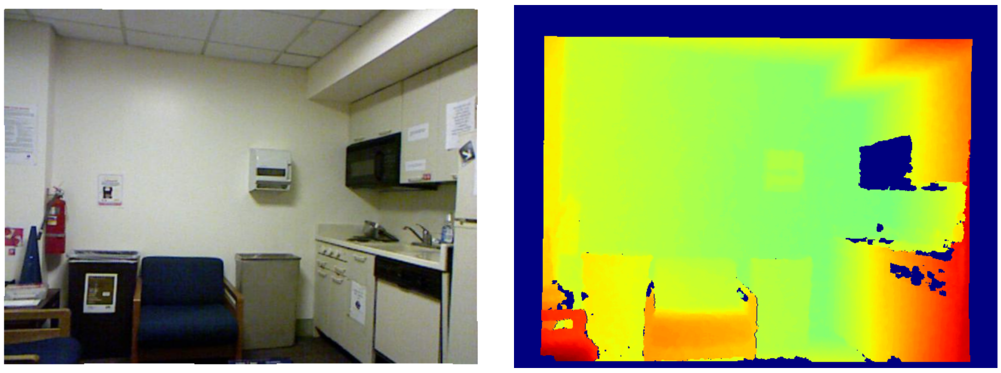


FIGURE 3.10: **Qualitative example from NYU v2.** From left to right, the reference image and the ground-truth depth map, colored as inverse depth.

Make3D. The Make3D dataset [262] contains single images and their ground-truth depth maps acquired using a laser scanner device. The dataset contains training and testing splits with 400 and 134 samples respectively, and it is mostly used for attesting generalization capability of monocular models. Images, which depict urban and natural areas, have a resolution of 2272×1704 pixels while depth maps have a notably smaller resolution (55×305). Figure 3.11 illustrates a qualitative example of the dataset taken from the original paper [262].

UnrealStereo4K. The UnrealStereo4K is a realistic synthetic high-res (3840×2160)

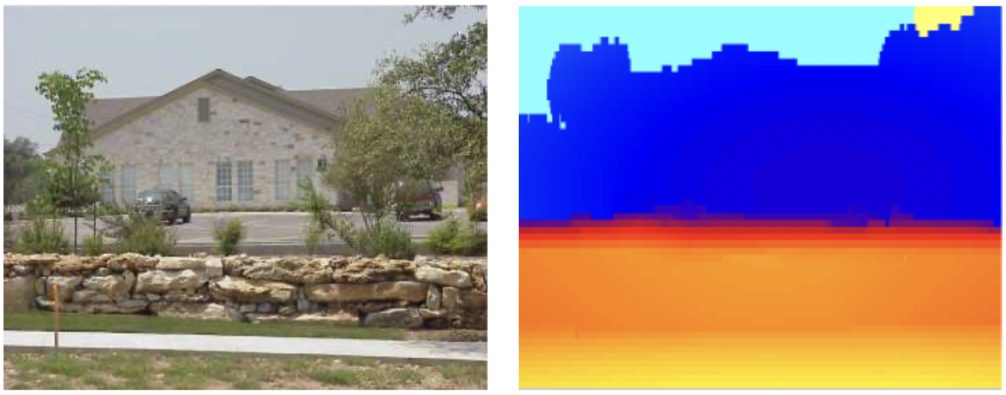


FIGURE 3.11: **Qualitative example from Make3D.** From left to right, the reference image and its corresponding ground-truth depth map (colored in the log scale, where close, mid and far distances are encoded in yellow, red and blue respectively).

stereo dataset with available ground-truth disparities. The dataset contains 7720, 80 and 200 in-domain pairs respectively for training, validation and test. Furthermore, to evaluate the generalization ability, the authors also provide an out-of-domain test set by rendering 200 stereo pairs from an unseen scene. A sample from the dataset is depicted in Figure 3.12.

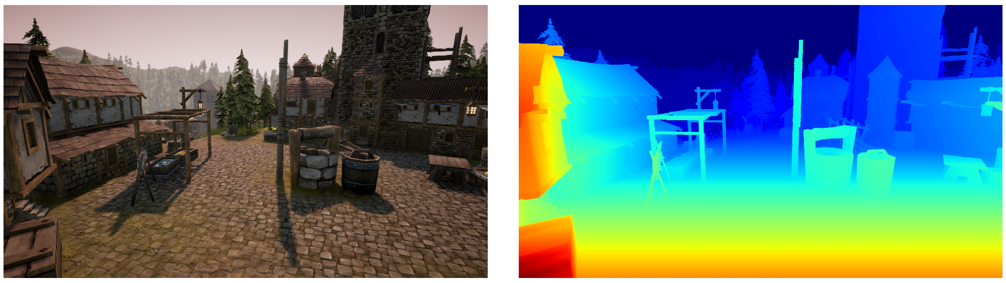


FIGURE 3.12: **Qualitative example from UnrealStereo4k.** From left to right, the reference image and the ground-truth disparity map.

Sintel. The MPI Sintel [37] is a synthetic dataset that depicts naturalistic video sequences with optical flow, motion boundaries, unmatched regions and disparity ground-truth labels. Image resolution is 1024 × 436. The training split counts about 1040 images with different lightning conditions. Clean and Final are the most adopted splits. In Figure 3.13 we illustrate an example of Sintel for optical flow.

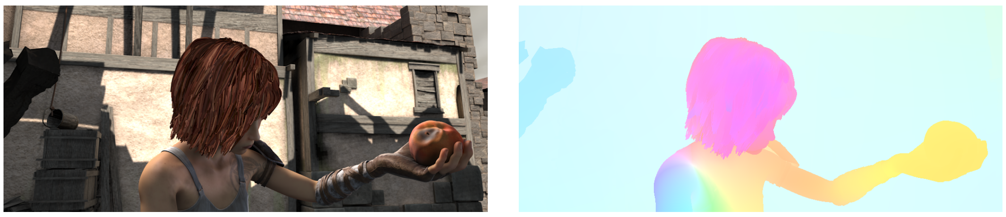


FIGURE 3.13: **Qualitative example from Sintel.** From left to right, the reference image and the ground-truth optical flow map.

FlyingChairs. FlyingChairs is a popular synthetic dataset used to train optical flow models. It contains 22,232 images of chairs moving according to 2D displacement vectors over random backgrounds sampled from Flickr. Image size is

512×384. We report an example of FlyingChairs in Figure 3.14.

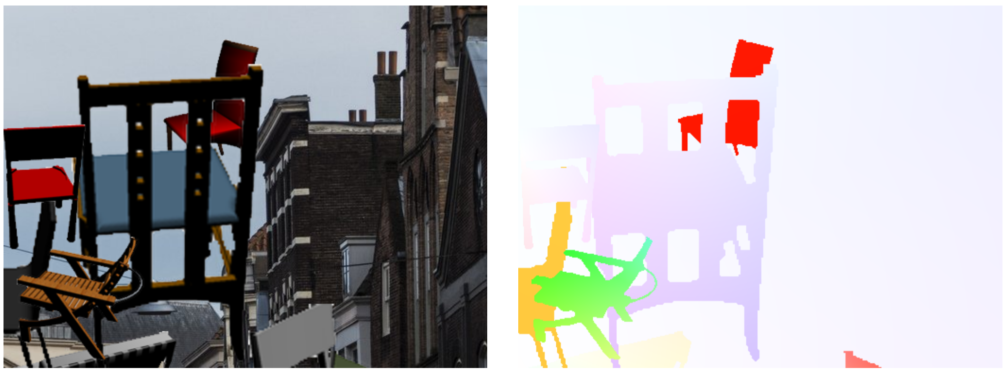


FIGURE 3.14: **Qualitative example from FlyingChairs.** From left to right, the reference image and the ground-truth optical flow map.

3.2 Metrics

Once a model has been trained, we are interested in computing its accuracy and errors in a quantitative way. Therefore, it is common to evaluate models exploiting ground-truth measurements as target values. Depending on the task, we might have different scores. In the following, we report the principal metrics used in monocular depth estimation, stereo, optical flow, confidence estimation and image classification and segmentation. Some metrics are the lower the better $\downarrow$, while others the higher the better $\uparrow$.

3.2.1 Monocular depth metrics

Popular metrics adopted in the monocular setting have been proposed in [65]. These metrics are computed in the depth space.

Abs Rel. $\downarrow$ It measures the difference of predictions with respect their corresponding ground-truth values (marked with *), normalized by ground-truth. The metric is averaged on valid points N, i.e. the set of points for which the ground-truth y^* is available.

$$\text{Abs Rel} = \frac{1}{|N|} \sum_{y \in N} \frac{|y - y^*|}{y^*} \tag{3.1}$$

Sq Rel. $\downarrow$ Similar to the Abs Rel, it computes the squared error instead the absolute one, thus assigning more importance to large errors.

$$\text{Sq Rel} = \frac{1}{|N|} \sum_{y \in N} \frac{(y - y^*)^2}{y^*} \tag{3.2}$$

RMSE. $\downarrow$ The Root Mean Squared Error (RMSE) measures how much the predictions fit the expected results. To obtain it, we have to take the mean of the squares of the residuals, and then to apply the square root.

$$\text{RMSE} = \sqrt{\frac{1}{|N|} \sum_{y \in N} (y - y^*)^2} \tag{3.3}$$

log RMSE. ↓ It is similar to the RMSE, but it computes the discrepancy when depths are in the log space.

$$\text{log RMSE} = \sqrt{\frac{1}{|N|} \sum_{y \in N} (\ln y - \ln y^*)^2} \tag{3.4}$$

Threshold. ↑ It measures the relative frequency of predicted depths for which a peculiar ratio δ is lower than a threshold ϵ. Popular choices for ϵ are 1.25, 1.25^2 and 1.25^3.

$$\delta < \epsilon = \frac{1}{|N|} \sum_{y \in N} \max(\frac{y}{y^*}, \frac{y^*}{y}) < \epsilon \tag{3.5}$$

Notice that, when testing multiple predictions, such as when our testing set is made of many images or we aim to test a video sequence and not over a single frame, the final score for each metric is computed taking the average of single evaluations. Moreover, since the metrics are computed in the depth space, if a model predicts inverse depths or disparities, first we have to move and align the predictions to the depth space, and then we can compute the metrics. Many methods that predict inverse depth require ground-truth values even during the alignment phase, to compute the scale in the depth space [80, 365] or a scale and shift alignment in the inverse depth domain [245], while methods that leverage stereo pairs at training time [79, 302] generally need only few parameters of the setting (e.g., the baseline and the focal length of the camera to turn the disparity into a depth).

3.2.2 Stereo matching metrics

Popular metrics to evaluate the quality of predicted disparity maps are EPE, D1 and BAD. These metrics are computed on the set N, i.e. the set of pixels for which ground-truth and predictions are both valid. Moreover, it is quite common to compute these metrics separately on different regions of the image. For instance, on KITTI [196] error metrics are measured both in *Noc* or *All*, respectively only in non-occluded areas and in all the pixels for which a valid ground-truth is available.

EPE. ↓ The End-Point-Error (EPE) measures the average discrepancy, in pixel units, between each predicted disparity value y and the correspondent ground-truth value y^*.

$$\text{EPE} = \frac{1}{|N|} \sum_{y \in N} |y - y^*| \tag{3.6}$$

D1. ↓ The D1 error, introduced in [196], measures the percentage of pixels for which the predicted disparity has a discrepancy with respect to the ground-truth higher than two thresholds, t_1 and t_2. Specifically, a pixel is considered invalid if its absolute disparity error is lower or equal than t_1 and if the ratio between this error and the ground-truth is larger or equal than t_2.

$$\begin{aligned} \text{D1} &= \tfrac{100}{|N|} \textstyle\sum_{y \in N} f(y, y^*) \\ f(y, y^*) &= \begin{cases} 1 & \text{if } |y - y^*| \geq t_1 \text{ and } \frac{|y-y^*|}{y^*} \geq t_2, \\ 0 & \text{otherwise} \end{cases}. \end{aligned} \tag{3.7}$$

where t_1 and t_2 are in general set to 3 and 0.05 respectively. In general, the metric takes into account all the valid pixels (D1-all), however KITTI [196] benchmark provides the score also for foreground objects (D1-fg) and for the background (D1-bg).

BAD. ↓ The BAD metric measures the percentage of pixels have an EPE error larger than a threshold ϵ. Generally, BAD is reported as BADϵ (e.g. BAD2).

$$\begin{aligned} &\text{BAD}\epsilon = \tfrac{100}{|N|} \textstyle\sum_{y \in N} b(y, y^*, \epsilon) \\ &b(y, y^*, \epsilon) = \begin{cases} 1 & \text{if } |y - y^*| \geq \epsilon, \\ 0 & \text{otherwise} \end{cases}. \end{aligned} \tag{3.8}$$

SEE. ↓ The Soft Edge Error (SEE) has been proposed in [46] to evaluate the disparity error for pixels around edge boundaries. In fact, edges are hard to predict for stereo networks and at the same time they count a little subset of the pixels in the image. Since popular metrics as EPE take into account all the valid pixels, methods that predict bad edges show similar performance to strategies that are instead reliable near edges, if they are accurate elsewhere.

$$\begin{aligned} &SEE_k(y, y^*) = \tfrac{1}{|E|} \textstyle\sum_{y \in E} se_k(y, y^*), \\ &se_k(y, y^*) = min|y - y^*_j| \quad j \in N_k(y), \end{aligned} \tag{3.9}$$

where $E = Edge(y^*)$ is the set of points in the edge boundaries and $N_k(y)$ denotes the local $k \times k$ neighbourhood of point y. Notice that when $k = 1$ we obtain the EPE measured on the edge.

3.2.3 Optical flow metrics

EPE. ↓ Similarly to stereo, EPE is measured also for optical flow. Given the ground-truth flow vector p^* and the predicted one p for each valid pixel, the EPE is computed as:

$$\text{EPE} = \frac{1}{|N|} \sum_{p \in N} \|p - p^*\|_2 \tag{3.10}$$

with $|N|$ the number of valid pixels. Notice that flow vectors contain two terms (i.e. a horizontal and a vertical component), while disparity in stereo is a scalar.

Fl. ↓ This metric, introduced in [196], is the corresponding of $D1$ for optical flow. It computes the percentage of pixels with absolute error larger or equal than t_1 and with the ratio between the absolute error and the ground-truth value larger or equal than t_2, with t_1 and t_2 two thresholds.

$$\begin{aligned} &\text{Fl} = \tfrac{100}{|N|} \textstyle\sum_{p \in N} f(p, p^*), \\ &f(p, p^*) = \begin{cases} 1 & \text{if } k(p, p^*) \geq t_1 \quad \text{and} \quad \frac{k(p,p^*)}{m(p^*)} \geq t_2, \\ 0 & \text{otherwise} \end{cases}, \\ &k(p, p^*) = \|p - p^*\|_2, \\ &m(p^*) = \sqrt{p_x^{*2} + p_y^{*2}} \end{aligned} \tag{3.11}$$

where p^*_x and p^*_y are the horizontal and vertical components of the ground-truth flow p^*. These two components are used to compute the magnitude m of the flow vector. In general we have $t_1 = 3$ and $t_2 = 0.05$.

>3. $\downarrow$ This metric is similar to Fl, in fact it computes the percentage of pixels with absolute error larger or equal than 3 pixels.

$$\begin{aligned} > 3 &= \tfrac{100}{|N|} \textstyle\sum_{p \in N} f(p, p^*), \\ f(p, p^*) &= \begin{cases} 1 & \text{if } k(p, p^*) \geq 3 \\ 0 & \text{otherwise} \end{cases}, \\ k(p, p^*) &= \|p - p^*\|_2, \end{aligned} \tag{3.12}$$

3.2.4 Confidence metrics.

AUSE. $\downarrow$ Also adopted in [119], the Area Under the Sparsification Error (AUSE) measures the uncertainty of the model. Given an error metric ϵ, first we rank all the points in order of descending uncertainty. Then, we progressively sample a certain percentage of points, and we compute ϵ on the remaining values. If the uncertainty measure is effective, the curve should shrink, since errors are removed at each iteration. The ideal case (i.e. having an *oracle*), instead, consists in the curve obtained by sorting the points in order of descending magnitude of ϵ. The AUSE measures the distance between the real and the ideal estimator, and it is computed as the area under the curve given by the difference between the estimated and optimal sparsifications.

AURG. $\uparrow$ The Area Under the Random Gain (AURG) is computed in a similar manner to AUSE, but instead of the ideal curve we have to adopt the curve generated by a random estimator. This latter curve is obtained by randomly removing points at each iteration. This measure estimates the gain of the uncertainty method compared to do not estimate uncertainty at all.

3.2.5 Classification and segmentation metrics

IoU. $\uparrow$ The Intersection over Union is a popular metric to evaluate image segmentation models. It is computed as:

$$\text{IoU} = \frac{TP}{TP + FN + FP} \tag{3.13}$$

where TP, FN and FP are respectively the true positive, the false negative and the false positive predictions of the model.

Precision. $\uparrow$ The Precision computes the ratio of correctly classified observations to the number of predicted positive cases.

$$P = \frac{TP}{TP + FP} \tag{3.14}$$

Recall. $\uparrow$ The metric computes the ratio of correctly classified observations to the actual number of positive cases.

$$R = \frac{TP}{TP + FN} \tag{3.15}$$

F1 score. $\uparrow$ The F1 score measures how good and complete are the predictions of the model. It is defined as the harmonic mean between the precision P and the recall R.

$$F1 = \frac{2 \cdot P \cdot R}{P + R} \tag{3.16}$$

mIoU class. ↑ The mean Intersection over Union over classes computes the IoU of each class and averages the result. This metric is widely adopted in semantic segmentation and it is computed as:

$$\text{mIoU class} = \frac{1}{n_{cl}} \sum_i \frac{n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}} \tag{3.17}$$

where n_{ij} indicates the number of pixels of class i predicted to belong to class j, n_{cl} is the number of classes and t_i is the total number of ground-truth pixels of class i. Notice that $n_{ii} = TP_i$, $t_i = TP_i + FN_i$ and $\sum_j n_{ji} = TP_i + FP_i$, where TP_i, FN_i and FP_i are TP, FN and FP computed for the i-th class.

mIoU cat. ↑ This metric is similar to mIoU class, but it takes into account categories instead of single classes. In particular, related classes are clustered together, such as car, truck, bus and bicycle are clustered into the same vehicle category.

Pixel Acc. ↑ The Pixel Accuracy is the relative frequency of pixels with correctly predicted labels. It is computed as:

$$\text{Pixel Acc.} = \frac{\sum_i n_{ii}}{\sum_i t_i} \tag{3.18}$$

Mean Acc. ↑ This metric computes the mean of the accuracy of each class.

$$\text{Mean Acc.} = \frac{1}{n_{cl}} \sum_i \frac{n_{ii}}{t_i} \tag{3.19}$$

f.w. IoU. ↑ The Frequency Weighted Intersection over Union extends mIOU to tackle class imbalance. It computes the weighted mean of the classes instead of the arithmetic average.

$$\text{f.w. IoU} = \frac{1}{\sum_k t_k} \sum_i \frac{t_i n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}} \tag{3.20}$$

3.3 Baseline models

We now discuss monocular depth models that are frequently used as baselines in this work.

MonoDepth. Proposed in [79], MonoDepth is a monocular depth model trained in a self-supervised fashion using stereo pairs. An initial encoder, which can have a ResNet50 [98] or a VGG-like [276] structure, extracts features while a decoder is in charge of predicting two disparity maps aligned with the left and right views of a virtual stereo pair: the left disparity agrees with the real input image, while the other map fits an unseen right image, not available in practice with a monocular setup. This allows applying additional consistency terms at training time, as the left-right consistency check. The VGG-like network contains 31 million trainable parameters, while the ResNet variant has 48 million parameters.

MonoDepth2. This network, presented in [80], introduces some novelties respect to MonoDepth. First, the proposed training scheme does not leverage only stereo pairs, as in MonoDepth, but also monocular videos. These two modalities can be also combined, obtaining a training procedure that leverages stereo pairs with monocular video frames simultaneously. Second, an effective auto-mask strategy is in charge of removing pixels that break the assumption of a moving camera in a stationary scene. Third, the authors introduce some loss innovations, such as the *per-pixel minimum reprojection*, that make the training procedure more effective.

In terms of architecture, the network has an encoder-decoder structure as in MonoDepth. The encoder, that can be a ResNet18 or a ResNet50 [98], shrinks the input and then a decoder module predicts a depth map at the original resolution. At each level in the decoder, 3×3 convolutions with skip connections are performed, followed by a 3×3 convolution layer in charge of depth estimation. The resulting network can predict depth at different scales, The model with the ResNet18 as encoder contains about 14.8 million trainable parameters, while the other 34.6 million parameters. Both the encoders can be trained from scratch, but they can also start from the already available ImageNet pre-training [58] to improve the speed of convergence.

MiDaS. Proposed by Ranftl et al. [245], MiDaS is a convolutional neural network trained to predict inverse depth maps for a large set of different data, such as 3D movies or stereo videos. The authors provide two models, characterized by different encoders and thus by different complexity: the large model, counting about 105 million parameters, and the small one, with 21 million parameters, referred to as MiDaS and MiDaS small. While the latter runs at a high framerate even on mobile devices (MiDaS small v2.1 runs at 30 FPS on an Apple iPhone 11), the former is more accurate.

DPT. The Dense Prediction Transformer (DPT) [244] represents the state-of-the-art solution for monocular depth estimation in the wild. It is based on a Visual Transformer (ViT [60]) backbone that takes a set of flattened embeddings (tokens) from non-overlapping patches of the image and processes them through a set of transformer layers. Then, a convolutional decoder reassembles the tokens into an image-like feature representation at different resolutions that are progressively fused. Finally, a dedicated head predicts the task-specific output at the original input resolution. Eventually, the tokens can be source using a CNN backbone: in this case, the model is called ViT-Hybrid, and exploits a ResNet50 [98]. DPT and DPT-Hybrid count nearly 344 and 123 million of trainable parameters respectively.

FastDepth. Proposed by Wofk et al. [326], this network can infer depth predictions at 178 fps with an NVIDIA Jetson TX2 GPU. This notable speed is the result of design choices, hardware-specific optimizations and pruning [342] strategies. The encoder is a MobileNet [108], 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.

Chapter 4

Boosting monocular depth networks with stereo supervision

This chapter presents monocular Residual Matching (shorten, MonoResMatch), a novel end-to-end architecture trained to estimate accurate (inverse) depth maps from single images. On the one hand, the accuracy is achieved by the peculiar design of the network, which mimics a stereo setup. Specifically, the reference input image is mapped in the features space, and from this representation the model estimates a first disparity outcome. We then synthesize features aligned with a *virtual* right image by warping the original features with the estimated disparity. A final refinement module performs stereo matching between the real and synthesized representations. On the other hand, to boost the performance we also leverage traditional knowledge from classical stereo methods to obtain proxy labels at training time. We will show that, despite the presence of outliers in proxy maps (in fact, these labels are far from being perfect ground-truth labels), optimizing the model according to this paradigm results in a superior accuracy compared to self-supervision with image reconstruction. Since the training pipeline requires disparity labels and not only images, the proposed method is not self-supervised as [79, 80]. However, these labels are easily computed from the same raw images used by these methods using off-the-shelf strategies, therefore we refer to this setup as weakly supervised. The content of this chapter is based on Learning monocular depth estimation infusing traditional stereo knowledge [302].

4.1 Monocular Residual Matching

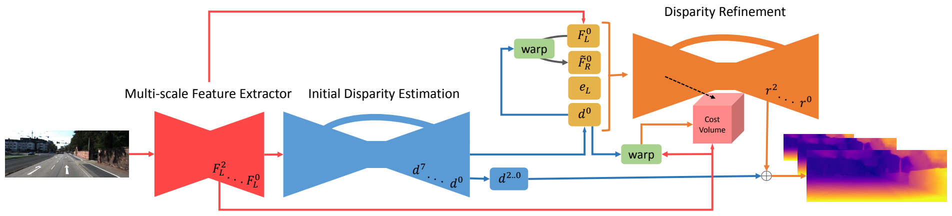


FIGURE 4.1: **Illustration of MonoResMatch architecture.** Given one input image, the multi-scale feature extractor (in red) generates high-level representations in the first stage. The initial disparity estimator (in blue) yields multi-scale disparity maps aligned with the left and right frames of a *virtual* stereo pair. The disparity refinement module (in orange) is in charge of refining the initial left disparity relying on features computed in the first stage, disparities generated in the second stage, matching costs between high-dimensional features F_L^0 extracted from input and synthetic $\tilde{F}_R^0$ from a *virtual* right viewpoint, together with absolute error e_L between F_L^0 and back-warped $\tilde{F}_R^0$.

In this section, we describe in detail the proposed MonoResMatch architecture designed to infer an accurate and dense depth map in a weakly-supervised manner from a single image. Figure 4.1 recaps the three key components of our network at test time. First, a multi-scale feature extractor takes as input a single raw image and computes deep learnable representations at different scales from quarter resolution F_L^2 to full-resolution F_L^0 in order to toughen the network to ambiguities in photometric appearance. Second, deep high-dimensional features at input image resolution are processed to estimate, through an hourglass structure with skip-connections, multi-scale inverse depth (i.e., disparity) maps aligned with the input and a *virtual* right view learned during training. By doing so, our network learns to emulate a binocular setup, thus allowing further processing in the stereo domain [181]. Third, a disparity refinement stage estimates residual corrections to the initial disparity. In particular, we use deep features from the first stage and back-warped features of the *virtual* right image to construct a cost volume that stores the stereo matching costs using a correlation layer [191].

Our entire architecture is trained from scratch in an end-to-end manner, while SVS [181] by training its two main components, Deep3D [332] and DispNetC [191], on image synthesis and disparity estimation tasks separately (with the latter requiring additional, supervised depth labels from synthetic imagery [191]).

Extensive experimental results prove that MonoResMatch enables much more accurate estimations compared to SVS and other state-of-the-art approaches.

4.1.1 Multi-scale feature extractor

Inspired by [165], given one input image I we generate deep representations using layers of convolutional filters. In particular, the first 2-stride layer convolves I with 64 learnable filters of size 7×7 followed by a second 2-stride convolutional layer composed of 128 filters with kernel size 4×4. Two deconvolutional blocks, with stride 2 and 4, are deployed to upsample features from lower-spatial resolution to full input resolution producing 32 features maps each. A 1×1 convolutional layer with stride 1 further processes upsampled representations.

4.1.2 Initial disparity estimation

Given the features extracted by the first module, this component is in charge of estimating an initial disparity map. In particular, an encoder-decoder architecture inspired by DispNet processes deep features at quarter resolution from the multi-scale feature extractor (i.e., *conv2*) and outputs disparity maps at different scales, specifically from $\frac{1}{128}$ to full-resolution. Each down-sampling module, composed of two convolutional blocks with stride 2 and 1 each, produces a growing number of extracted features, respectively 64, 128, 256, 512, 1024, and each convolutional layer uses 3×3 kernels followed by ReLU non-linearities. Differently from DispNet, which computes matching costs in the early part of this stage using features from the left and right images of a stereo pair, our architecture lacks such necessary information required to compute a cost volume since it processes a single input image. Thus, no 1-D correlation layer can be imposed to encode geometrical constraints in this stage of our network. Then, upsampling modules are deployed to enrich feature representations through skip-connections and to extract two disparity maps, aligned respectively with the input frame and a *virtual* viewpoint on its right as in [79]. This process is carried out at each scale using 1-stride convolutional layers with kernel size 3×3.

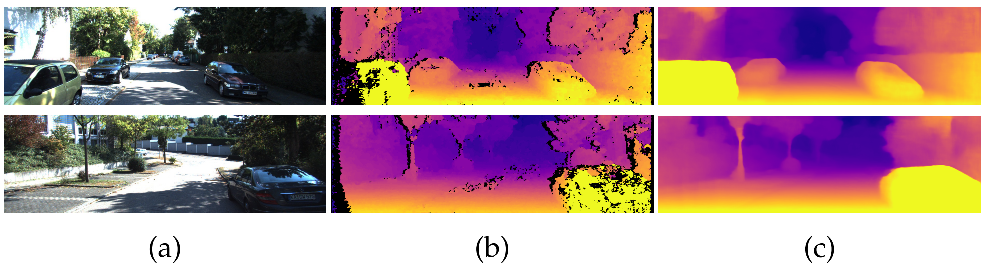


FIGURE 4.2: **Examples of proxy labels computed by SGM.** Given the source image (a), the network exploits the SGM supervision filtered with left-right consistency check (b) in order to train MonoResMatch to estimate the final disparity map (c). No post-processing from [79] is performed on (c) in this example.

4.1.3 Disparity refinement

Given an initial estimate of the disparity at each scale obtained in the second part of the network, often characterized by errors at depth discontinuities and occluded regions, this stage predicts corresponding multi-scale residual signals [98] by a few stacked nonlinear layers that are then used to compute the final left-view aligned disparity map. This strategy allows us to simplify the end-to-end learning process of the entire network. Moreover, motivated by [181], we believe that geometrical constraints can play a central role in boosting the final depth accuracy. For this reason, we embed matching costs in feature space computed employing a horizontal correlation layer, typically deployed in deep stereo algorithms. To this end, we rely on the right-view disparity map computed previously to generate right-view features $\tilde{F}_R^0$ from the left ones F_L^0 using a differentiable bilinear sampler [122]. The network is also fed with error e_L, i.e. the absolute difference between left and *virtual* right features at input resolution, with the latter back-warped at the same coordinates of the former, as in [165].

We point out once more that, differently from [181], our architecture produces both a synthetic right view, i.e. its features representation, and computes the final disparity map following stereo rationale. This makes MonoResMatch a single end-to-end architecture, effectively performing stereo out of a single input view rather than the combination of two models (i.e., Deep3D [332] and DispNetC [191] for the two tasks outlined) trained independently as in [181]. Moreover, exhaustive experiments will highlight the superior accuracy achieved by our weakly-supervised end-to-end approach.

4.1.4 Training loss

In order to train our multi-stage architecture, we define the total loss as a sum of two main contributions, a $\mathcal{L}_{init}$ term from the initial disparity estimation module and a $\mathcal{L}_{ref}$ term from the disparity refinement stage. Following [80], we embrace the idea to up-sample the predicted low-resolution disparity maps to the full input resolution and then compute the corresponding signals. This simple strategy is designed to force the inverse depth estimation to reproduce the same objective at each scale, thus leading to much better outcomes. In particular, we obtain the final training loss as:

$$\mathcal{L}_{total} = \sum_{s=1}^{n_i} \mathcal{L}_{init} + \sum_{s=1}^{n_r} \mathcal{L}_{ref} \tag{4.1}$$

where s indicates the output resolution, n_i and n_r the numbers of considered scales during loss computation, while $\mathcal{L}_{init}$ and $\mathcal{L}_{ref}$ are formalised as:

$$\begin{aligned}\mathcal{L}_{init} = &\alpha_{ap}(\mathcal{L}^l_{ap} + \mathcal{L}^r_{ap}) + \alpha_{ds}(\mathcal{L}^l_{ds} + \mathcal{L}^r_{ds}) \\ &+ \alpha_{ps}(\mathcal{L}^l_{ps} + \mathcal{L}^r_{ps})\end{aligned} \tag{4.2}$$

$$\mathcal{L}_{ref} = \alpha_{ap}\mathcal{L}^l_{ap} + \alpha_{ds}\mathcal{L}^l_{ds} + \alpha_{ps}\mathcal{L}^l_{ps} \tag{4.3}$$

where $\mathcal{L}_{ap}$ is an image reconstruction loss, $\mathcal{L}_{ds}$ is a smoothness term and $\mathcal{L}_{ps}$ is a proxy-supervised loss. Each term contains both the left and right components for the initial disparity estimator, and the left components only for the refinement stage.

Image reconstruction loss. A linear combination of $L1$ loss and *structural similarity measure* (SSIM) [319] encodes the quality of the reconstructed image $\tilde{I}$ with respect to the original image I for each pixel i, j:

$$\mathcal{L}_{ap} = \frac{1}{N}\sum_{i,j} \alpha \frac{1 - SSIM(I_{ij}, \tilde{I}_{ij})}{2} + (1-\alpha)|I_{ij} - \tilde{I}_{ij}| \tag{4.4}$$

Following [79], we set $\alpha = 0.85$ and use a SSIM with 3×3 block filter.

Disparity smoothness loss. This cost encourages the predicted disparity to be locally smooth, and it has been employed also in other works [80, 79]. Disparity gradients are weighted by an edge-aware term from image domain:

$$\mathcal{L}_{ds} = \frac{1}{N}\sum_{i,j} |\partial_x D_{ij}| e^{-|\partial_x I_{ij}|} + |\partial_y D_{ij}| e^{-|\partial_y I_{ij}|} \tag{4.5}$$

,

where D_{ij} is the disparity for the pixel at coordinates i, j.

Proxy-supervised loss. When ground-truth labels are not available, we can use off-the-shelf methods to distill pseudo labels for training purposes [143, 321]. Specifically, given the proxy disparity maps obtained by a conventional stereo algorithm, detailed in Section 4.2, we coach the network using reverse Huber (berHu) loss [216]:

$$\mathcal{L}_{ps} = \frac{1}{N}\sum_{i,j} berHu(D_{ij}, D^{st}_{ij}, c_{ij}) \tag{4.6}$$

$$berHu(D_{ij}, D^{st}_{ij}, c_{ij}) = \begin{cases} |D_{ij} - D^{st}_{ij}| & \text{if } |D_{ij} - D^{st}_{ij}| \le c_{ij} \\ \frac{|D_{ij} - D^{st}_{ij}|^2 - c^2_{ij}}{2c_{ij}} & \text{otherwise} \end{cases} \tag{4.7}$$

where D_{ij} and D^{st}_{ij} are, respectively, the predicted disparity and the proxy annotation for pixel at the coordinates i, j of the image, while c is adaptively set for each pixel as $c_{ij} = 0.2 \max |D_{ij} - D^{st}_{ij}|$.

4.2 Proxy labels distillation

To generate accurate proxy labels, we use the popular SGM algorithm [101], a fast yet effective solution to infer depth from a rectified stereo pair without training. In our implementation, initial matching costs are computed for each pixel p and disparity hypothesis d applying a 9×7 census transform and computing Hamming distance on pixel strings. Then, scanline optimization along eight different paths refines the initial cost volume following the equation 2.2, and the final winner-takes-all strategy, applied to each pixel, produces the final disparity map D.

Although SGM generates quite accurate disparity labels, outliers may affect the training of a depth model negatively, as noticed by Tonioni et al.[300]. They applied a learned confidence measure [233] to filter out erroneous labels when computing the loss. Differently, we run a non-learning based left-right consistency check to detect outliers. Purposely, by extracting both disparity maps D^L and D^R with SGM, respectively for the left and right images, we apply the following criteria to invalidate (i.e., set to -1) pixels having different disparities across the two maps:

$$D(p) = \begin{cases} D(p) & \text{if } |D^L(p) - D^R(p - D^L(p))| \leq \varepsilon \\ -1 & \text{otherwise} \end{cases} \tag{4.8}$$

The left-right consistency check is a simple strategy that removes many wrong disparity assignments, mostly near depth discontinuities, without needing any training that would be required by [300]. Therefore, our proxy labels generation process does not rely at all on ground-truth depth labels. Figure 4.2 shows an example of distilled labels (b), where black pixels correspond to outliers filtered out by left-right consistency. Although some of them persist, we can notice how they do not affect the final prediction by the trained network and how our proposal can recover accurate disparity values in occluded regions on the left side of the image (c).

4.3 Experimental results

In this section we present exhaustive evaluations of MonoResMatch on various training/testing configurations, showing that our proposal consistently outperforms self-supervised state-of-the-art approaches. As standard in this field, we assess the performance of monocular depth estimation techniques following the protocol by Eigen et al.[65], i.e. extracting data from the KITTI [196] dataset and using sparse LiDAR measurements as ground-truth for evaluation. Additionally, we also perform an exhaustive ablation study proving that proxy supervision from SGM algorithm and effective architectural choices enable our strategy to improve predicted depth map accuracy by a large margin.

For training purposes we used Cityscapes followed by KITTI. We refer to these two training sets as Cityscapes (CS) and Eigen KITTI split (K) from now on. More details about the training procedure are given in Appendix A.

4.3.1 Ablation study

In this section we examine the impact of i) proxy-supervision from SGM and ii) the different components of MonoResMatch. The outcomes of these experiments, conducted on the Eigen split, are collected in Table 4.1.

Proxy-supervised loss analysis. We train MonoDepth framework by Godard et al.[79] from scratch adding our proxy-loss, then we compare the obtained model with the original one, as well as with the more effective strategy used by 3Net [237]. It is worth noticing that the models have a similar complexity in terms of number of trainable parameters: Godard et al.[79](ResNet50), Poggi et al.[237](ResNet50) and the proposed MonoResMatch have 48.0, 48.0 and 42.5 million of parameters respectively. Furthermore, all the models perform the post-processing technique defined in [79], which helps to boost the performance.

Method	Supervision		Train set	Abs Rel	Sq Rel	RMSE	log RMSE	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
	Image	SGM								
Godard et al.[79] ResNet50	✓		K	0.128	1.038	5.355	0.223	0.833	0.939	0.972
Poggi et al.[237] ResNet50	✓		K	0.126	0.961	5.205	0.220	0.835	0.941	0.974
MonoResMatch	✓		K	0.116	0.986	5.098	0.214	0.847	0.939	0.972
MonoResMatch	✓	✓	K	**0.111**	**0.867**	**4.714**	**0.199**	**0.864**	**0.954**	**0.979**
Godard et al.[79] ResNet50	✓		CS,K	0.114	0.898	4.935	0.206	0.861	0.949	0.976
Poggi et al.[237] ResNet50	✓		CS,K	0.111	0.849	4.822	0.202	0.865	0.952	0.978
Godard et al.[79] ResNet50	✓	✓	CS,K	0.110	0.822	4.675	0.199	0.862	0.953	0.980
MonoResMatch (no-refinement)	✓	✓	CS,K	0.107	0.781	4.588	0.195	0.869	0.957	0.980
MonoResMatch (no-corr)	✓	✓	CS,K	0.104	0.766	4.553	0.192	0.875	0.958	0.980
MonoResMatch (no-pp)	✓	✓	CS,K	0.098	0.711	4.433	0.189	0.888	0.960	0.980
MonoResMatch	✓	✓	CS,K	**0.096**	**0.673**	**4.351**	**0.184**	**0.890**	**0.961**	**0.981**

TABLE 4.1: **Ablation studies on Eigen split [65].** Maximum depth value is set to 80m. All networks perform the post-processing as in [79] unless otherwise specified (no-pp entry). All the methods employ stereo pairs at training time.

We can observe that proxy-loss enables a more accurate MonoDepth model (row 3) compared to [79], moreover it also outperforms virtual trinocular supervision proposed in [237], attaining better metrics with respect of both, but $\delta < 1.25$ for 3Net. Specifically, by recalling Figure 4.2, the proxy distillation couples well with a cropping strategy, solving well-known issues for stereo supervision such as disparity ramps on the left border.

Component analysis. Still referring to Table 4.1, we evaluate different configurations of our framework by ablating the key modules peculiar to our architecture. First, we train MonoResMatch on K without proxy supervision (row 3) to highlight that our architecture already outperforms [79] (row 1). Training on CS+K with proxy labels, we can notice how without any refinement module (*no-refinement*), our framework already outperforms the proxy-supervised ResNet50 model of Godard et al.[79]. Adding the disparity refinement component without encoding any matching relationship (*no-corr*) enables small improvements, becoming much larger on most metrics when a correlation layer is introduced (*no-pp*) to process real and synthesized features as to resemble stereo matching. Finally, post-processing as in [79] (row 11) still ameliorates all scores, although the larger contribution is given by the correlation-based refinement module, as perceived by comparing *no-refinement* and *no-pp* entries. Finally, by comparing rows 4 and 11 we can also perceive the impact given by CS pretraining on our full model.

4.3.2 Comparison with self-supervised frameworks

Having studied in detail the contribution of both MonoResMatch architecture and proxy supervision, we compare our framework with state-of-the-art self-supervised approaches for monocular depth estimation. Table 4.2 collects results obtained evaluating different models on the aforementioned Eigen split [65]. In this evaluation, we consider only competitors trained without *any* supervision from ground-truth labels (e.g., synthetic datasets [191]) involved in *any* phase of the training process [181, 89]. We refer to methods using monocular supervision (*Seq*), binocular (*Stereo*) or both (*Seq+Stereo*). Most methods are trained on CS and K, except Yang et al.[341] that leverages on different sub-splits of K. From the table, we can notice that MonoResMatch outperforms all of them significantly.

To compete with methods exploiting supervision from dense synthetic ground-truth [191], we run additional experiments using very few annotated samples from KITTI as in [181, 89], for a more fair comparison. Table 4.3 collects the outcome of

Method	PP	Sup	Train set	Abs Rel	Sq Rel	RMSE	log RMSE	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Zou et al.[367]		Seq	CS,K	0.146	1.182	5.215	0.213	0.818	0.943	0.978
Mahjourian et al.[187]		Seq	CS,K	0.159	1.231	5.912	0.243	0.784	0.923	0.970
Yin et al.[347] GeoNet ResNet50		Seq	CS,K	0.153	1.328	5.737	0.232	0.802	0.934	0.972
Wang et al.[314]		Seq	CS,K	0.148	1.187	5.496	0.226	0.812	0.938	0.975
Poggi et al.[230] PyD-Net (200)		Stereo	CS,K	0.146	1.291	5.907	0.245	0.801	0.926	0.967
Godard et al.[79] ResNet50	✓	Stereo	CS,K	0.114	0.898	4.935	0.206	0.861	0.949	0.976
Poggi et al.[237] 3Net ResNet50	✓	Stereo	CS,K	0.111	0.849	4.822	0.202	0.865	0.952	0.978
Pilzer et al.[226] (Teacher)		Stereo	CS,K	0.098	0.831	4.656	0.202	0.882	0.948	0.973
Yang et al.[341]	✓	Seq+Stereo	K_o, K_r, K_o	0.097	0.734	4.442	0.187	0.888	0.958	0.980
MonoResMatch	✓	Stereo	CS,K	**0.096**	**0.673**	**4.351**	**0.184**	**0.890**	**0.961**	**0.981**

TABLE 4.2: **Quantitative evaluation on Eigen split et al.[65].** Maximum depth value is set to 80m. PP entries use the post-processing of [79]. K_o, K_r, K_o are splits from K, defined in [341]. Best results are shown in bold.

these experiments according to different degrees of supervision, in particular using accurate ground-truth labels from the KITTI 2015 training split (200-acrt) or different amounts of samples from K with LiDAR measurements, respectively 100, 200, 500 and 700 as proposed in [181, 89], running only 5k iterations for each configuration. We point out that MonoResMatch, on direct comparisons to methods trained with the same amount of labeled images, consistently achieves better scores, with rare exceptions. Moreover, we highlight in red for each metric the best score among all the considered configurations, figuring out that MonoResMatch trained with 200-acrt plus 500 samples from K attains the best accuracy on all metrics. This fact points out the high effectiveness of the proposed architecture, able to outperform state-of-the-art techniques [181, 89] trained with much more supervised data (i.e., more than 30k stereo pairs from [191] and pre-trained weights from ImageNet). Leveraging on the traditional SGM algorithm instead of a deep stereo network as in [89] for proxy-supervision ensures a faster and easier to handle training procedure.

Method	Supervision					Abs Rel	Sq Rel	RMSE	log RMSE	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
	200-acrt	100	200	500	700							
Luo et al.[181]	✓					0.101	0.673	4.425	**0.176**	-	-	-
MonoResMatch	✓					**0.089**	**0.575**	**4.186**	0.181	0.897	0.964	0.982
Luo et al.[181]	✓		✓			0.100	0.670	4.437	0.192	0.882	0.958	0.979
MonoResMatch	✓		✓			**0.096**	**0.573**	**3.950**	**0.168**	**0.897**	**0.968**	**0.987**
Luo et al.[181]	✓			✓		0.094	0.635	4.275	0.179	0.889	0.964	0.984
MonoResMatch	✓			✓		**0.093**	**0.567**	**3.914**	**0.165**	**0.901**	**0.969**	**0.987**
Luo et al.[181]	✓				✓	**0.094**	0.626	4.252	0.177	0.891	0.965	0.984
MonoResMatch	✓				✓	0.095	**0.567**	**3.942**	**0.166**	**0.899**	**0.969**	**0.987**
Guo et al.[89]		✓				**0.096**	0.641	4.095	**0.168**	0.892	0.967	0.986
MonoResMatch		✓				0.098	**0.597**	**3.973**	0.169	**0.895**	**0.968**	**0.987**

TABLE 4.3: **Comparison between methods supervised by few annotated samples.** Experimental results on the Eigen split [65], with maximum depth value set to 80m. The label 200-acrt means using the 200 pairs of KITTI 2015 with ground-truth labels, while 100, 200, 500, 700 indicate how many LiDAR-annotated pairs have been included for training. Best results in direct comparisons are shown in bold, best overall scores are in red, consistently attained by MonoResMatch

4.3.3 Comparison with Depth Hints

We now compare the proposed MonoResMatch with Depth Hints, a contemporary work proposed by Watson et al.[321]. Similarly to MonoResMatch, Depth Hints exploits stereo pairs at training time to generate proxy labels with SGM [101]. However, it differs from MonoResMatch for the following reasons. First, the formulation of Depth Hints allows to use even monocular sequences in addition to stereo pairs at training time, as in [80]; second, the proxy depth labels are not applied to every pixel, but selectively ameliorate only the points for which the reprojection loss can be improved. Table 4.4 reports the comparison between the two methods on the Eigen split [65] when only stereo pairs are employed for training. For each model, we report the million of trainable parameters (MP), the image resolution used at training time (W×H) and the dataset used for the pre-training dataset (PT), when required. Moreover, all the configurations perform the post-processing of [79]. We can notice that, despite the differences in terms of image resolution, model complexity and pre-training datasets, the two methods show similar performance when using their best configurations.

Method	MP	PT	W×H	Abs Rel	Sq Rel	RMSE	log RMSE	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Depth Hints (ResNet50)	34.6		1024 × 320	0.112	0.857	4.807	0.203	0.861	0.952	0.978
Depth Hints	14.8	I	1024 × 320	0.099	0.723	4.445	0.187	0.886	0.962	0.981
Depth Hints (ResNet50)	34.6	I	1024 × 320	**0.096**	0.710	4.393	0.185	**0.890**	**0.962**	**0.981**
MonoResMatch	42.5		640 × 192	0.111	0.867	4.714	0.199	0.864	0.954	0.979
MonoResMatch	42.5	CS	640 × 192	**0.096**	**0.673**	**4.351**	**0.184**	**0.890**	0.961	**0.981**

TABLE 4.4: **Comparison with Depth Hints.** We compare MonoResMatch with the competitor work Depth Hints [321] on the Eigen split [65]. All the methods exploit the same post-processing strategy, but they can be eventually pre-trained (PT) on ImageNet [58] (I) or Cityscapes [55] (CS). Column MP reports the number of trainable parameters (in millions), while W×H the training resolution.

4.3.4 Performance on single view stereo estimation

Finally, we further compare MonoResMatch directly with Single View Stereo (SVS) by Luo et al.[181], being both driven by the same rationale. We fine-tuned MonoResMatch on the KITTI 2015 training set as in Table 4.3 and submitted to the online stereo benchmark [196] as in [181]. Table 4.5 compares MonoResMatch with SVS and other techniques evaluated in [181], respectively MonoDepth [79] and OpenCV Block-Matching (OCV-BM). D1 scores, defined in Chapter 3.2, represent the percentages of pixels having a disparity error larger than 3 or 5% of the ground-truth value on different portions of the image, respectively background (bg), foreground (fg) or its entirety (all). We can observe from the table a margin larger than 3% on D1-bg and near to 1% for D1-fg, resumed in a total reduction of 2.72%. This outcome supports once more the superiority of MonoResMatch, although SVS is trained on many, synthetic images with ground-truth [191]. Finally, Figure 4.3 depicts qualitative examples retrieved from the KITTI online benchmark.

Method	PP	D1-bg	D1-fg	D1-all
MonoDepth [79]	✓	27.00	28.24	27.21
OCV-BM		24.29	30.13	25.27
SVS [181]		25.18	20.77	24.44
MonoResMatch	✓	**22.10**	**19.81**	**21.72**

TABLE 4.5: **Quantitative results on the test set of the KITTI 2015 Stereo Benchmark [196].** Percentage of pixels having error larger than 3 or 5% of the ground-truth. Best results are shown in bold.

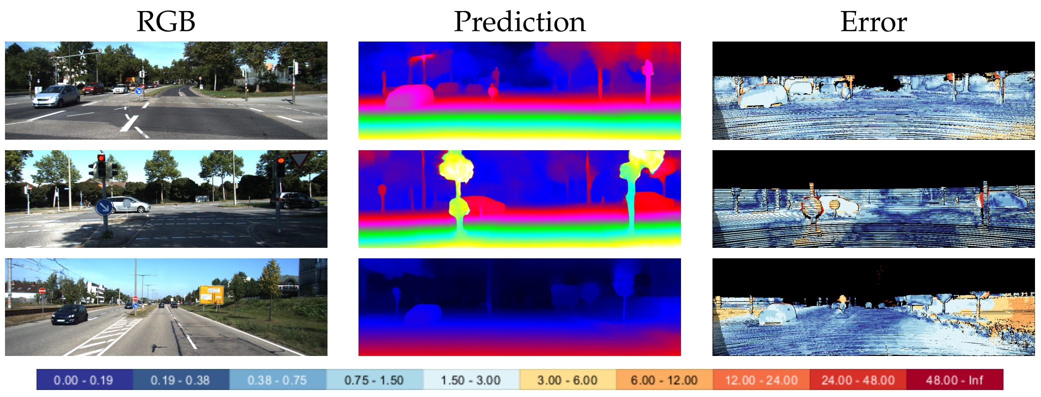


FIGURE 4.3: **Online evaluation of MonoResMatch.** From left to right the input image, the predicted depth and the errors with respect to ground-truth. The last line reports the color code used to display the seriousness of the shortcomings (same of [196])

4.4 Conclusions

In this chapter we have seen in details MonoResMatch, a novel framework for accurate monocular depth estimation. It combines i) specific design choices to tackle depth-from-mono in analogy to stereo matching, thanks to a correlation-based refinement module and ii) a more robust training leveraging the proxy ground-truth labels generated through a traditional (i.e. non-learning based) algorithm such as SGM. Although the proposed strategy is not self-supervised as [80, 79] due to need of proxy labels, these labels are easily extracted from the same data adopted by these methods. For this reason, proxy-label distillation can be thought of as a peculiar data pre-processing for the raw stereo dataset.

In contrast to state-of-the-art models [181, 89, 341], our architecture is elegantly trained in an end-to-end manner. Through exhaustive experiments, we prove that plugging proxy-supervision at training time leads to more accurate networks. This claim is also supported by parallel works, such as [321].

Chapter 5

Monocular depth estimation on low-power devices

In the previous chapter, we have discussed MonoResMatch, a monocular model that can infer accurate depth maps. Unfortunately, the other side of the coin is that we pay the accuracy in terms of model's size and efficiency: the model counts about 42.5 million parameters and performs a large number of operations, preventing the real-time execution even when using high-performance workstations equipped with powerful GPUs. A similar reasoning applies to MonoDepth [79], which counts about 31 million parameters in its VGG-like variant and 48 millions in the larger ResNet50 version, and many other models. Furthermore, in real applications we would like to deploy our models on portable devices, such as a Raspberry Pi3, because they consume less energy, they are cheaper and require much less space than high-performance GPUs. In [230], Poggi et al. show that monocular depth inference on embedded devices is possible: the proposed model, called PyD-Net, can work in real-time on standard CPUs and at ∼2 FPS on low-power devices such as a Raspberry Pi 3 with an accuracy comparable to much more complex architectures [79]. This chapter extends that work under two aspects: **i)** to further shrink the gap in accuracy with more complex models, we followed recent methods [80] as well as the weakly-supervised training strategy with proxy labels described in Chapter 4 and **ii)** we re-designed the architecture of the network. The proposed network PyD-Net2 has ∼35% the complexity of the original PyD-Net and runs in real-time on a broad range of embedded devices, allowing for sharp depth estimates at an unrivaled speed.

The contents of this chapter are based on the article Real-time Self-Supervised Monocular Depth Estimation Without GPU [234].

5.1 PyD-Net architectures

This section describes the main modules composing our compact architectures derived from our preliminary work [230]. The basic blocks are the pyramidal feature extractors, in place of the traditional encoder, and multiple depth decoders processing outputs at different scales. Starting from these basic blocks, the two architectures depicted in Figures 5.2 are built, respectively referred to as PyD-Net [230] and PyD-Net2. From now on, depth maps estimated at the half, quarter and eighth scale respectively are referred as at H, Q and E scales.

Stereo self-supervision is used to train our networks, because of the advantages with respect to monocular sequences. Specifically, monocular sequences suffer inherently in presence of independently moving objects and the camera pose is frequently unknown, while the stereo rig acquires the two view points simultaneously

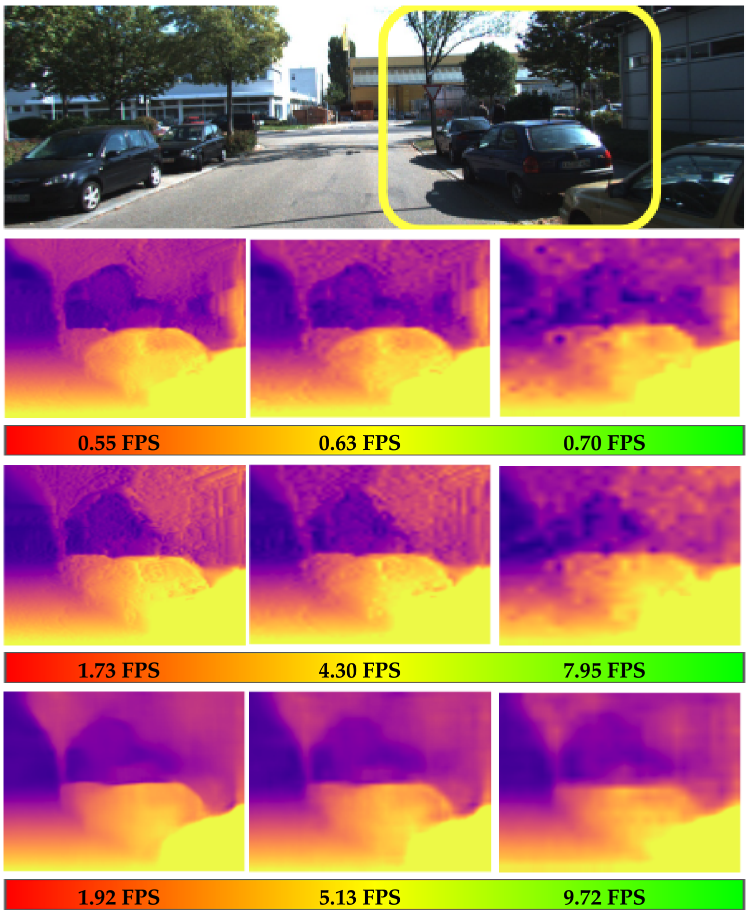


FIGURE 5.1: **Single image depth estimation on low-powered hardware.** From top to bottom, results by MonoDepth [79] and our compact architectures PyD-Net[230] and PyD-Net2. From left to right, results at different output scales (half, quarter and eighth). For each model and scale, frame rates achieved on the ARM Cortex A57 embedded CPU of the NVIDIA Jetson Nano are repoted.

and the relative camera pose is given after the calibration.

For PyD-Net [230], the model is trained as in its original version according to the methodology proposed in [79]. Differently, for PyD-Net2 modern proxy-supervision techniques are used, relying on a mixture of image synthesis losses [79] and proxy depth labels obtained using a traditional stereo algorithm such as SGM [101].

5.1.1 Pyramidal features extractor

In the proposed architectures, a small encoder is deployed to extract multi-scale features from the input image. According to the desired number of scales, a different number of consecutive convolutional blocks compose the whole features extractor. Each block is composed of two convolutional layers, respectively with stride 2 and 1. The very first block of the sequence produces the first level of the pyramid, namely L1, at half resolution, followed by more blocks until reaching the desired lowest scale. Every convolutional layers uses 3×3 kernels to keep memory and runtime requirements low, and each block extracts a small number of features. Considering a pyramid with 6 levels, from L1 to L6 as for PyD-Net [230] from half to $\frac{1}{64}$ resolution, every block extracts respectively 16, 32, 64, 96, 128 and 192 features. These numbers

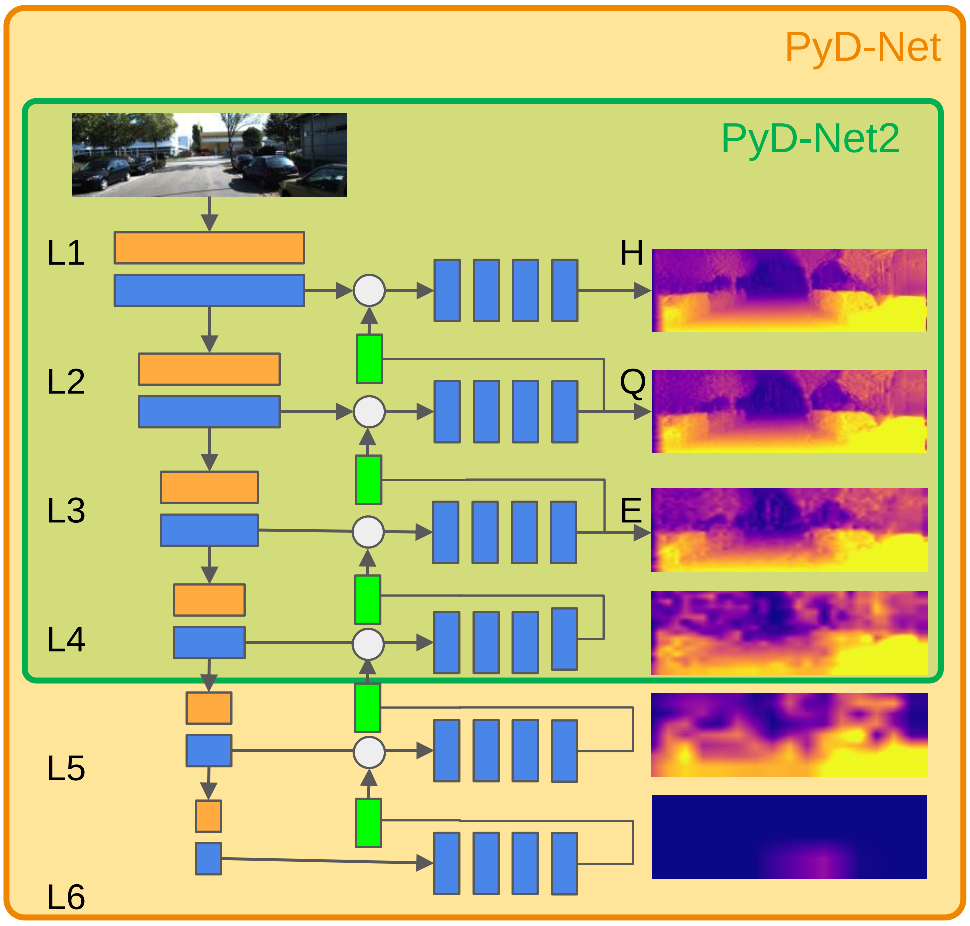


FIGURE 5.2: **PyD-Net architectures.** The original PyD-Net model proposed in [230] (orange) counts about 1.9M parameters, the novel PyD-Net2 design (green) only 700K. Blue and orange blocks are 3×3 convolutions with stride 1 and 2, respectively. Green blocks are upsampling layers.

are much smaller compared to the amounts extracted for instance by VGG [276], that are 32, 64, 128, 256, 512, 512. Each layer is followed by leaky ReLUs with $\alpha = 0.2$.

5.1.2 Depth decoders

Once multi-scale features have been extracted by the pyramidal encoder, a decoder is in charge of depth estimation at each scale. In order to keep the network as compact as possible, each decoder consists of four 3×3 convolutional layers, extracting 96, 64, 32 and 8 feature maps. A leaky ReLU follows each layer except for the last one as in the pyramidal encoder. From the output of each decoder, two inverse depth maps (i.e., disparity maps) are extracted, respectively from the point of view of left and right images belonging to stereo pairs used at training time as in [79], by applying different activations depending on which self-supervised losses are optimized during training. This strategy will be discussed later in detail.

Moreover, features extracted by the last layer are upsampled by employing 2×2 transposed convolutions applying a stride of 2, in order to increase the spatial scale by the same factor. Then, such upsampled features are concatenated with those at the same scale extracted by the pyramidal encoder and forward to the decoder in charge of depth estimation. This design choice allows for a coarse-to-fine strategy, enabling to estimate depth maps with varying degree of accuracy according to the pyramid level, up to half of the original input resolution as in [230], where full resolution estimation was not implemented to reduce runtime and memory requirements (with minor impact on accuracy). Thus, as proposed in [230], one can trade accuracy for speed by early stopping computation at lower scales. The effect of such a strategy will be studied in detail, showing good practices to soften the price to pay regarding accuracy and at the same time achieve advantages concerning runtime and memory footprint.

5.1.3 Self-supervision and proxies

For training, binocular stereo pairs are leveraged in place of ground-truth labels for supervision. Specifically, for PyD-Net the same strategy proposed in [79] is followed, using image synthesis, smoothness constraints and left-right consistency between the two depth maps inferred by the network as supervisory signals. On the other hand, PyD-Net2 is trained by means of a combination of image synthesis signals, smoothness constraints and proxy ground-truth labels inferred by deploying SGM. The following loss function, deployed for depth maps computed at each scale, encompasses both approaches:

$$\begin{aligned}\mathcal{L}_{\text{s}} =&\alpha_{\text{ap}}(\mathcal{L}^{l}_{\text{ap}} + \mathcal{L}^{r}_{\text{ap}}) + \alpha_{\text{ds}}(\mathcal{L}^{l}_{\text{ds}} + \mathcal{L}^{r}_{\text{ds}})+\\ &\alpha_{\text{lr}}(\mathcal{L}^{l}_{\text{lr}} + \mathcal{L}^{r}_{\text{lr}}) + \alpha_{\text{px}}(\mathcal{L}^{l}_{\text{px}} + \mathcal{L}^{r}_{\text{px}})\end{aligned} \tag{5.1}$$

resulting in a weighted sum of image reprojection $\mathcal{L}_{\text{ap}}$, smoothness $\mathcal{L}_{\text{ds}}$, left-right consistency $\mathcal{L}_{\text{lr}}$ and proxy supervision $\mathcal{L}_{\text{px}}$ losses, weighted by respective α weights. The overall loss formulation is similar to the one used to train MonoResMatch in Chapter 4.2: in fact, all the terms are the same with the only exception for the consistency term α_{lr}. Each of these terms will be explained in detail next, assuming as the reference the left image (tag ^{l}). Nonetheless, according to (5.1) the same loss functions can be computed assuming as reference the right image (tag ^{r}), where each estimator infers two disparity maps as in MonoDepth [79].

Image reprojection loss. The image appearance loss proposed in [79] is used. We already presented this term in Chapter 4.4: for any pixel at coordinates *i,j*, it measures the reconstruction error between the original image I^l and the reconstructed image $\tilde{I}^l$, obtained by warping the right image I^r according to estimated disparity. The loss uses a weighted combination between SSIM [319] and L1 terms. As we did in Chapter 4, we set this α weight to be 0.85.

Smoothness loss. This term, presented in Chapter 4.5, discourages depth discontinuities according to L1 penalty, unless a gradient δI occurs on the image.

Left-right consistency loss. It enforces coherence between predicted left D^l and right D^r disparity maps for the pixel at coordinates i, j. This term has been employed also in [79].

$$\mathcal{L}^{l}_{\text{lr}} = \frac{1}{N}\sum_{i,j}|D^{l}_{ij} - D^{r}_{ij+D^{l}_{ij}}| \tag{5.2}$$

Proxy supervision loss. The proxy supervision loss, introduced in Chapter 4.6, measures the discrepancy between the predicted disparity value and the proxy label for each pixel. As in Chapter 4, we adopt the reverse Huber [216] for this purpose, and we exploit SGM [101] as stereo proxy generator, following the strategy described in Chapter 4.2. Also in this case, we apply 9×7 windows for census transform and we set the left-right difference threshold ε of equation 4.8 to 3. Pixels breaking the left-right consistency are set to -1 and thus discarded from loss computation. We adopt the SGM implementation from [280] to produce labels.

5.1.4 PyD-Net variants

Built upon the design strategies mentioned above, two main architectures and training methodologies are outlined.

PyD-Net. The original PyD-Net architecture [230], framed in orange in Figure 5.2, and recalled in this paragraph. It deploys pyramid levels L1 to L6 and six estimators to produce depth maps at scales from half to $\frac{1}{64}$. Input image resolution is set to 512×256. The overall training loss involves image appearance, disparity smoothness and left-right consistency check, setting weights in (5.1) respectively to $\alpha_{\text{ap}} = 1$, $\alpha_{\text{ds}} = \frac{0.1}{r}$ (with r being the downsampling factor at each scale), $\alpha_{\text{lr}} = 1$ and $\alpha_{\text{px}} = 0$. Losses are computed at each scale of the depth estimator. To do so, left and right images are downsampled to the six different scales and used to compute supervision signals. Note that, according to such strategy, each disparity needs to be multiplied by a scale factor when upsampled to full resolution, i.e. disparities at half resolution need to be multiplied by a scale factor of 2. The final layer of each estimator has a Sigmoid activation unit multiplied by a factor 0.3 as in [79]. To obtain disparities, the outputs are multiplied by the image width. PyD-Net counts 1.9 M parameters, about 6% compared to MonoDepth [79].

PyD-Net2. The model framed in green in Figure 5.2 proposed to improve accuracy, execution time and memory footprint. The main modification consists of a more compact architecture, deploying only a portion of the original pyramidal features extractor, respectively from L1 to L4. We argue that lowest PyD-Net levels learn too much coarse feature representations negligibly contributing to the higher resolution depth map estimation. Moreover, by removing such levels in PyD-Net2 we not only obtain a faster lightweight model but also improve the estimated depth accuracy compared to our previous model. Therefore, the resulting PyD-Net2 network counts only 700 K parameters, about 65% and 97.5% smaller compared to PyD-Net [230] and MonoDepth [79] respectively.

Concerning the overall training loss, weights for PyD-Net2 are set to $\alpha_{\text{ap}} = 1$, $\alpha_{\text{ds}} = 0.1$, $\alpha_{\text{lr}} = 0$ and $\alpha_{\text{px}} = 1$, thus adding proxy self-supervision. The left-right consistency check is turned off, because already encoded in the post-processing phase deployed by SGM to obtain proxy labels. Consequently, Sigmoid activations are replaced with ReLU.

Finally, additional strategies allowed to further improve accuracy:

- During the inference phase an input size of 640×192 is adopted, enabling to keep the original aspect ratio of KITTI images. This choice is particularly important to improve results on thin structures. Moreover, the network is trained on random crops rather than on re-sized full images in order to solve the disparity ramp issue that usually appears on the left side of the image due to the stereo occlusions. This strategy is effective because, by randomly cropping SGM disparity maps during training, the network exploits proxy supervision in regions where the re-projection losses would fail. More specifically, given the raw stereo images and disparity maps computed by SGM at the original 1280×384 KITTI resolution, random crops of 384×896 size are first extracted and then fed to the network after resizing to half the resolution 448×192, enabling the architecture to operate with lower spatial dimensions.

- The loss function is computed only for H, Q and E depth maps, avoiding optimization on the output at L4 scale. Moreover, all losses are computed by upsampling disparity maps rather than downsampling images and proxy labels, following [80]. This method allows us to improve lower scale estimates and dramatically reduce the loss in accuracy when early stopping our network. By doing so, outputs are upsampled to the original full image resolution, i.e. 896×384 crop size, and not to the 448×192 input size. This keeps all the information from the proxy labels and thus boosts accuracy, at the cost of a slower training procedure due to the much more considerable amount of signals computed that however does not affect the runtime during inference.

Aside from these two main models, additional configurations will be studied during the experiments. Respectively, a variant of the original PyD-Net trained with PyD-Net2 strategies, namely **PyD-Net++**, and a variant of PyD-Net2 processing 320×96 to pursue efficiency, namely **PyD-Net2-RT**.

5.2 Experimental results

In this section, an exhaustive evaluation concerning the different variants of the proposed pyramidal approach is reported, highlighting how the combination of architectural and training paradigm modifications allows to obtain significant improvements in depth estimation from a single input image compared to our original proposal [230] and other state-of-the-art solutions. Furthermore, a detailed analysis concerning runtime and memory footprint of the designed networks using hardware devices with different computing power resources is conducted, showing how it is possible to achieve real-time inference even on very low-power systems, whilst at the same time maintaining good accuracy in depth prediction. We refer to Appendix B for additional details about training procedures.

5.2.1 Competitors

First, the self/proxy-supervised networks selected for our experiments are introduced, divided into three categories.

High-complexity models. Huge networks processing large resolution images belong to this category, focusing on achieving state-of-the-art accuracy without caring about runtime and memory requirements. Among them, DepthHints [321] in its ResNet50 HR version and MonoResMatch are selected.

Medium-complexity models. Methods belonging to this category process images at lower resolution (typically, 512×256 or 640×192), thus requiring less memory and being faster. MonoDepth VGG [79], 3Net [237] and MonoDepth2 [80] are chosen as representative of this category[1].

Low-complexity models. These networks have very few amount of parameters (less than 2M), run fast and require a low amount of memory independently of the resolution. Our PyD-Net variants belong to this category, together with FastDepth [326] for which an exhaustive evaluation in terms of memory and runtime is reported, since proposed and tested for supervised depth estimation only.

[1]although high-complexity variants of MonoDepth (using ResNet50) and MonoDepth2 (running on 1024×320) exist, they perform worse than DepthHints and MonoResMatch, thus their lower-complexity variants are studied

However, some models (e.g. Depth Hints and MonoDepth2) have been developed using PyTorch [222] while others (e.g. 3Net and MonoResMatch) are in TensorFlow [189], and this may cause a discrepancy when measuring the runtime. For this reason, runtime performance have been evaluated after porting all the models to TensorFlow.

5.2.2 Ablation study

First, the impact of both the training protocol and architectural choices converting PyD-Net to PyD-Net2 is measured. Notice that, even if we follow some innovations from MonoDepth2 [80], such as the loss computation at full-res, we now compare against MonoDepth [79], which was the principal competitor of the original PyD-Net.

Method	Image Res.	SGM	Abs Rel	Sq Rel	RMSE	log RMSE	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
MonoDepth [79]	512×256		0.124	1.076	5.311	0.219	0.847	0.942	0.973
MonoDepth++	512×256	✓	0.123	1.137	5.093	0.218	0.854	0.946	0.974
PyD-Net[230]	512×256		0.146	1.291	5.907	0.245	0.801	0.926	0.967
PyD-Net[230]	640×192		0.141	1.270	5.741	0.238	0.815	0.930	0.968
PyD-Net++	640×192	✓	0.135	1.155	5.532	0.227	0.824	0.935	0.972
PyD-Net2	640×192	✓	0.127	1.059	5.259	0.218	0.834	0.942	0.974

TABLE 5.1: **Ablation study on KITTI [196] using the Eigen split [65].** All models have been trained on Cityscapes [55] and then fine-tuned on the Eigen training split of KITTI [65].

Table 5.1 shows the outcome of this study on the Eigen's split of KITTI. First, the MonoDepth architecture is trained with our protocol, dubbing the resulting model as MonoDepth++, to compare with the original model [79]. The same resolution, 512×256, is kept because of constraints by the VGG encoder, requiring images multiple of 128. According to the results, the updated training protocol using proxy labels and crops is beneficial for MonoDepth too. Then, various PyD-Net models are trained to assess how each modification impacts the final accuracy. By changing the input resolution, the results already improve. By adding proxy supervision by means of SGM further improves the accuracy, as well as replacing the original PyD-Net architecture with the more compact PyD-Net2 model. It is worth pointing out how the gap between PyD-Net and MonoDepth shrinks when moving to PyD-Net2 and MonoDepth++. In particular, the difference in terms of $\delta < 1.25$ is halved.

5.2.3 Evaluation on KITTI dataset

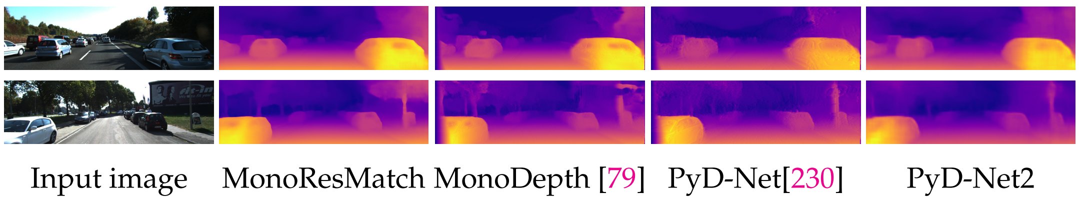


FIGURE 5.3: **Qualitative results on the Eigen split [65].** From left to right we show input image and depth estimations by, respectively, MonoResMatch, MonoDepth [79], PyD-Net[230], and PyD-Net2.

Method	Image Res.	Training	SGM	Pars.	Abs Rel	Sq Rel	RMSE	log RMSE	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$	FPS
DepthHints [321]	1024×320	I+K	✓	35M	**0.097**	0.733	4.445	**0.186**	**0.889**	**0.962**	**0.981**	36.23
MonoResMatch	1280×384	CS+K	✓	43M	0.098	**0.711**	**4.433**	0.189	0.888	0.960	0.980	12.53
MonoDepth [79]	512×256	CS+K		31M	0.124	1.076	5.311	0.219	0.847	0.942	0.973	77.20
3Net [237]	512×256	CS+K		48M	0.117	0.905	4.982	0.210	0.856	0.948	0.976	56.23
MonoDepth2 [80]	640×192	I+K		15M	0.109	0.873	4.960	0.209	0.864	0.948	0.975	133.51
PyD-Net[230]	512×256	CS+K		1.9M	0.146	1.291	5.907	0.245	0.801	0.926	0.967	203.96
PyD-Net2	640×192	CS+K	✓	**0.7M**	0.127	1.059	5.259	0.218	0.834	0.942	0.974	280.74
PyD-Net2-RT	320×96	CS+K	✓	**0.7M**	0.145	1.260	5.773	0.236	0.797	0.925	0.970	**370.92**

TABLE 5.2: **Evaluation on KITTI [196] using the Eigen split [65].** For training, K refers to KITTI dataset, CS to Cityscapes [55] and I to ImageNet pretraining [58]. We report networks in descending order of complexity from top to bottom, underlining best metrics in each group and showing in bold absolute winners. Frames per second measures on NVIDIA Titan Xp GPU.

In Table 5.2 our networks is compared with the competitors on the Eigen's split of the KITTI raw dataset.

As anticipated, PyD-Net2 significantly improves depth prediction accuracy compared to our original proposal [230], minimizing the gap with MonoDepth [79] and obtaining substantially equivalent results despite the number of parameters are drastically reduced by about 44 times. Despite this major achievement, it is worth observing that 3Net [237] achieves higher accuracy at the cost of a larger pool of parameters (48 M). Compared to PyD-Net2, it is increased by a factor of 68.5. MonoDepth2 further improves with respect to MonoDepth and 3Net with half the amount of parameters, thanks to the ImageNet pre-training [58] and several better practises during training. Yet, PyD-Net variants count about 8 and 21 times fewer parameters. This also translates into a performance gain in terms of runtime, as shown in the last column of Table 5.2. Indeed, PyD-Net variants are dramatically faster.

Although competitive with medium-complexity models, the gap with respect to state-of-the-art architectures such as DepthHints and MonoResMatch is higher. This gap shrinks when studying generalization, as reported next. Figure 5.3 presents some qualitative examples on Eigen split.

5.3 Generalization on Make3D dataset

In Table 5.3 the generalization performance on the Make3D dataset [262] is studied, following the evaluation proposed in [80], on a center crop of 2×1 ratio and without applying median scaling (not required when training on stereo pairs). In terms of generalization, the gap between high-complexity networks and our compact models shrinks. In particular, PyD-Net2 also results more effective than MonoResMatch on all metrics, highlighting that the margin between state-of-the-art huge models and our compact networks almost vanishes when moving to unseen environments.

Figure 5.4 shows some qualitative examples.

5.3.1 Accuracy-efficiency trade-off

In this section, an exhaustive study concerning the accuracy vs speed trade-off allowed by the early-stop strategy introduced in our previous work [230] is reported. In particular, given a network, depth maps are estimated at the full input resolution (F), as well as at half (H), quarter (Q) and eighth (E) scale. By early-stopping at any of the previous, intermediate scales, the layers following such output (usually, upsampling and convolutional layers) are cut-off from inference. The lower the scale, the higher the number of operations avoided, usually at the cost of some accuracy

Method	Training	SGM	Abs Rel	Sq Rel	RMSE	log RMSE
DepthHints [321]	I+K	✓	**0.350**	**3.385**	**8.242**	**0.200**
MonoResMatch [80]	CS+K	✓	0.375	4.072	8.859	0.213
MonoDepth [79]	CS+K		0.478	8.162	10.250	0.220
3Net [237]	CS+K		0.407	5.060	8.558	0.203
MonoDepth2 [80]	I+K		0.375	3.694	8.218	0.204
PyD-Net [230]	CS+K		0.510	9.106	10.538	0.225
PyD-Net2	CS+K	✓	0.375	3.930	8.551	0.210
PyD-Net2-RT	CS+K	✓	0.527	6.884	12.062	0.343

TABLE 5.3: **Generalization on the Make3D dataset [262].** No median rescaling [365] is performed.

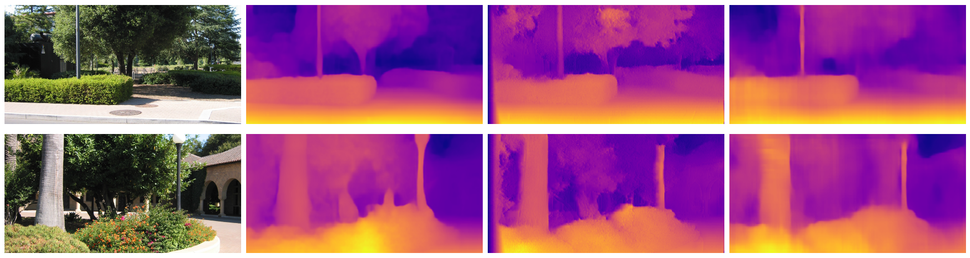


FIGURE 5.4: **Qualitative results Make3D [262].** From left to right we show input image and depth estimations respectively by MonoResMatch, MonoDepth [79] and PyD-Net2.

loss. Since reducing the input resolution generally allows to reduce the number of computations (as shown by PyD-Net2-RT in Table 5.2), the accuracy and speed of all the models trained and tested on images at half the original resolution is also reported. These variants are dubbed with “-RT" suffix, in analogy to PyD-Net2-RT.

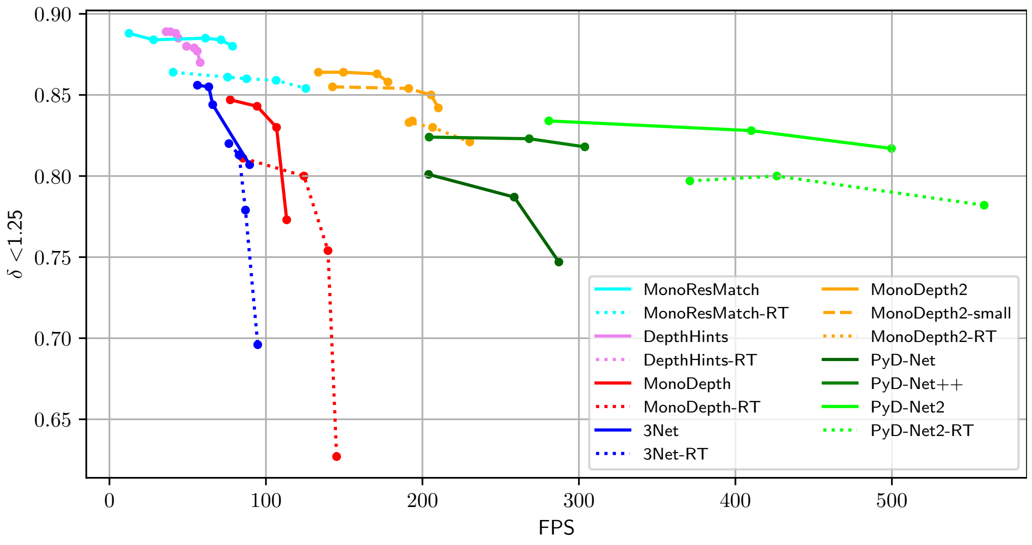
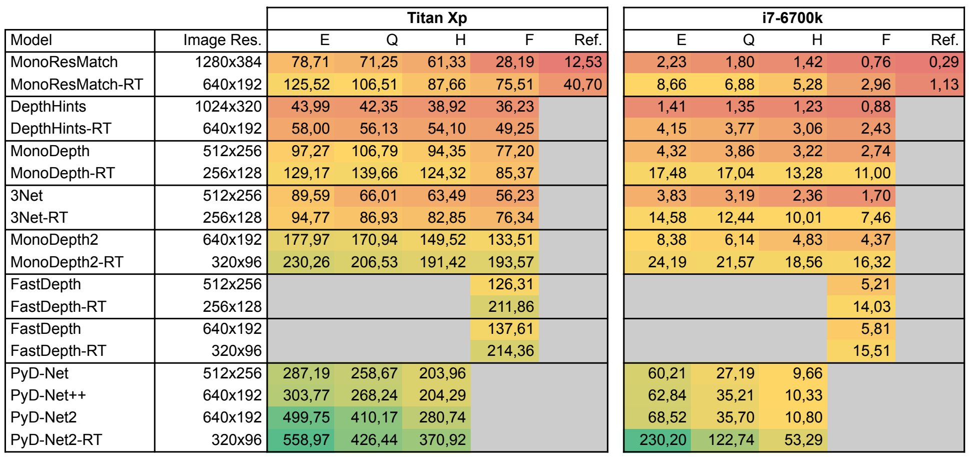


FIGURE 5.5: **Accuracy-speed trade-off.** We plot $\delta < 1.25$ computed by early-stopping each network at the available, intermediate scales (e.g. for MonoResMatch, from left to right, the plots correspond to Ref, F, H, Q and E). Runtime measured on NVIDIA Titan Xp.

Figure 5.5 plots the FPS (measured on NVIDIA Titan Xp) and $\delta < 1.25$ metrics achieved by different networks when early-stopping. For most networks, F, H, Q and E results are reported, for PyD-Net variants only H, Q and E because of their design. Finally, for MonoResMatch the results obtained by the full architecture involving the refinement subnetwork are reported as well (Ref.) together with F, H, Q and E obtained by cutting-off such module. It is worth noticing how DepthHints and MonoResMatch have negligible drops in accuracy, while not gaining much in terms of speed. The same behavior occurs with their RT variants, although MonoResMatch-RT suffers a higher accuracy drop, yet breaking the 100 FPS barrier at the two lowest scales. MonoDepth and 3Net have major drops in accuracy while MonoDepth2 results much more robust to stops at lower scales. These three do not gain much in terms of speed, although MonoDepth2 nearly reaches 200 FPS on the Titan Xp. MonoDepth2-small variant is also reported, trained on 416$\times$128 images and shown in the original paper [80], to highlight that it breaks the 200 FPS barrier at the lowest scale with minor accuracy loss. MonoDepth-RT and 3Net-RT do not gain much speed but suffer a dramatic loss in terms of depth metrics, while MonoDepth2-RT has a much more moderate drop and results faster than MonoDepth2-small.

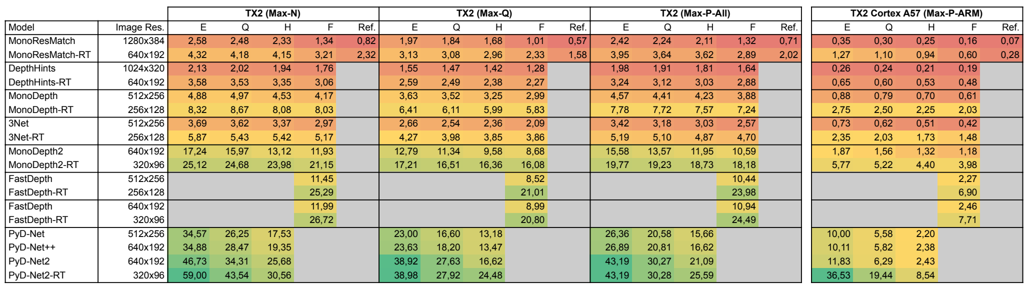
Finally, PyD-Net variants benefit much more in terms of efficiency gain. While the original PyD-Net suffers a noticeable loss in accuracy (although lower than the drops seen on MonoDepth), PyD-Net++ and PyD-Net2 show better robustness, with the latter achieving impressive gains in terms of speed (reaching almost 500 FPS). Such robustness, achieved by MonoDepth2, DepthHints and MonoResMatch as well, is ascribed to the fact that losses are computed on intermediate results upsampled to full resolution. Finally, PyD-Net2-RT further increases the efficiency, achieving nearly 400 FPS at the highest scale yet keeping about 0.80 $\delta < 1.25$.



		Titan Xp					i7-6700k				
Model	Image Res.	E	Q	H	F	Ref.	E	Q	H	F	Ref.
MonoResMatch	1280x384	78,71	71,25	61,33	28,19	12,53	2,23	1,80	1,42	0,76	0,29
MonoResMatch-RT	640x192	125,52	106,51	87,66	75,51	40,70	8,66	6,88	5,28	2,96	1,13
DepthHints	1024x320	43,99	42,35	38,92	36,23		1,41	1,35	1,23	0,88	
DepthHints-RT	640x192	58,00	56,13	54,10	49,25		4,15	3,77	3,06	2,43	
MonoDepth	512x256	97,27	106,79	94,35	77,20		4,32	3,86	3,22	2,74	
MonoDepth-RT	256x128	129,17	139,66	124,32	85,37		17,48	17,04	13,28	11,00	
3Net	512x256	89,59	66,01	63,49	56,23		3,83	3,19	2,36	1,70	
3Net-RT	256x128	94,77	86,93	82,85	76,34		14,58	12,44	10,01	7,46	
MonoDepth2	640x192	177,97	170,94	149,52	133,51		8,38	6,14	4,83	4,37	
MonoDepth2-RT	320x96	230,26	206,53	191,42	193,57		24,19	21,57	18,56	16,32	
FastDepth	512x256				126,31					5,21	
FastDepth-RT	256x128				211,86					14,03	
FastDepth	640x192				137,61					5,81	
FastDepth-RT	320x96				214,36					15,51	
PyD-Net	512x256	287,19	258,67	203,96			60,21	27,19	9,66		
PyD-Net++	640x192	303,77	268,24	204,29			62,84	35,21	10,33		
PyD-Net2	640x192	499,75	410,17	280,74			68,52	35,70	10,80		
PyD-Net2-RT	320x96	558,97	426,44	370,92			230,20	122,74	53,29		

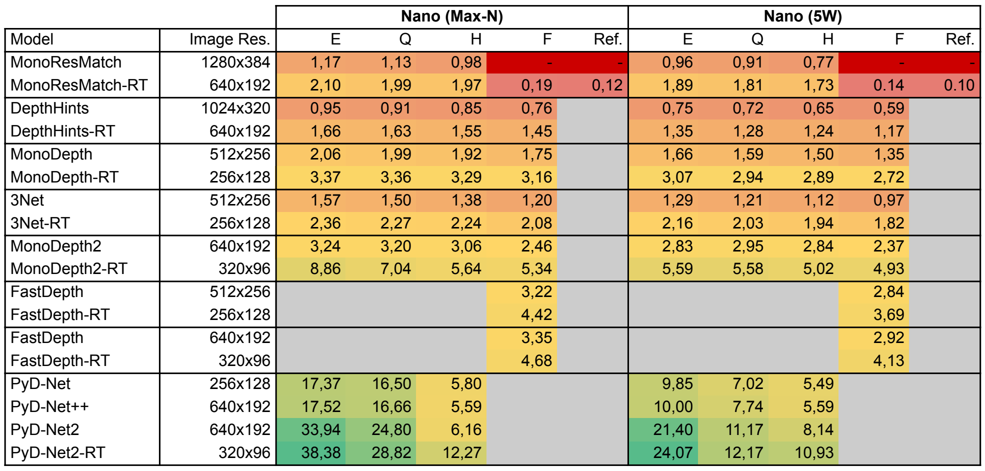
FIGURE 5.6: **Runtime analysis on NVIDIA Titan Xp GPU (left) and i7 CPU (right).** For each device, we report input resolution and FPS when varying the output scale.

PyD-Net models, particularly PyD-Net2, are more flexible in terms of accuracy-speed trade-off, either when training RT variants or when applying early-stopping. Moreover, the latter strategy results in the most flexible choice for any architecture – since it can be seamlessly applied to a pre-trained model, whereas RT variants i) require to train a model from scratch and ii) suffer significant accuracy drops. For instance, MonoResMatch operating at E is faster and more accurate than MonoResMatch-RT processing Ref. output, while MonoDepth and 3Net at Q outperform MonoDepth-RT and 3Net-RT at F. Although most networks easily break the 100 FPS barrier on NVIDIA Titan Xp thanks to early-stop or lower input resolution, the next section will show that only PyD-Net variants achieve real-time performance on embedded devices through this strategies.

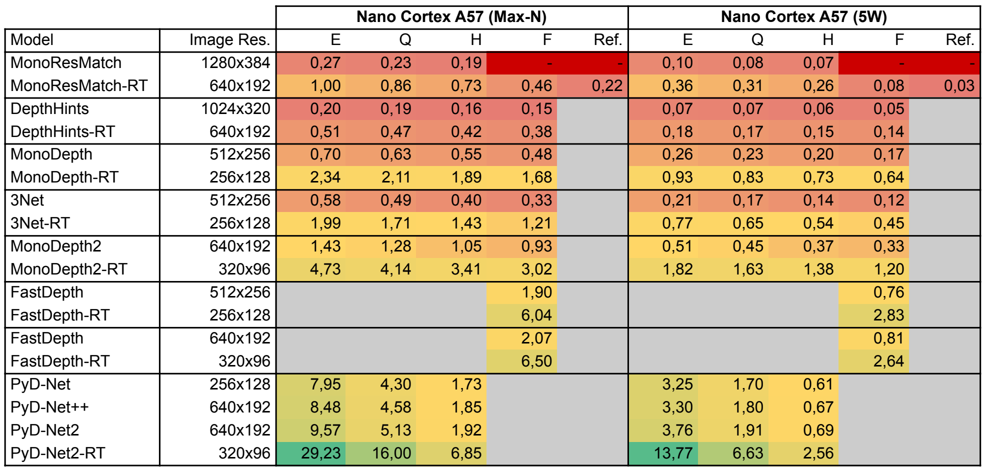


		TX2 (Max-N)					TX2 (Max-Q)					TX2 (Max-P-All)					TX2 Cortex A57 (Max-P-ARM)				
Model	Image Res.	E	Q	H	F	Ref.	E	Q	H	F	Ref.	E	Q	H	F	Ref.	E	Q	H	F	Ref.
MonoResMatch	1280x384	2,58	2,48	2,33	1,34	0,82	1,97	1,84	1,68	1,01	0,57	2,42	2,24	2,11	1,32	0,71	0,35	0,30	0,25	0,16	0,07
MonoResMatch-RT	640x192	4,32	4,18	4,15	3,21	2,32	3,13	3,08	2,96	2,33	1,58	3,95	3,64	3,62	2,89	2,02	1,27	1,10	0,94	0,60	0,28
DepthHints	1024x320	2,13	2,02	1,94	1,76		1,55	1,47	1,42	1,28		1,98	1,91	1,81	1,64		0,26	0,24	0,21	0,19	
DepthHints-RT	640x192	3,58	3,53	3,35	3,06		2,59	2,49	2,38	2,27		3,24	3,12	3,03	2,88		0,65	0,60	0,53	0,48	
MonoDepth	512x256	4,88	4,97	4,53	4,17		3,63	3,52	3,25	2,99		4,57	4,41	4,23	3,88		0,88	0,79	0,70	0,61	
MonoDepth-RT	256x128	8,32	8,67	8,08	8,03		6,41	6,11	5,99	5,83		7,78	7,72	7,57	7,24		2,75	2,50	2,25	2,03	
3Net	512x256	3,69	3,62	3,37	2,97		2,66	2,54	2,36	2,09		3,42	3,18	3,03	2,57		0,73	0,62	0,51	0,42	
3Net-RT	256x128	5,87	5,43	5,42	5,17		4,27	3,98	3,85	3,86		5,19	5,10	4,87	4,70		2,35	2,03	1,73	1,48	
MonoDepth2	640x192	17,24	15,97	13,12	11,93		12,79	11,34	9,58	8,68		15,58	13,57	11,95	10,59		1,87	1,56	1,32	1,18	
MonoDepth2-RT	320x96	25,12	24,68	23,98	21,15		17,21	16,51	16,36	16,08		19,77	19,23	18,73	18,18		5,77	5,22	4,40	3,98	
FastDepth	512x256				11,45					8,52					10,44					2,27	
FastDepth-RT	256x128				25,29					21,01					23,98					6,90	
FastDepth	640x192				11,99					8,99					10,94					2,46	
FastDepth-RT	320x96				26,72					20,80					24,49					7,71	
PyD-Net	512x256	34,57	26,25	17,53			23,00	16,60	13,18			26,36	20,58	15,66			10,00	5,58	2,20		
PyD-Net++	640x192	34,88	28,47	19,35			23,63	18,20	13,47			26,89	20,81	16,62			10,11	5,82	2,38		
PyD-Net2	640x192	46,73	34,31	25,68			38,92	27,63	16,62			43,19	30,27	21,09			11,83	6,29	2,43		
PyD-Net2-RT	320x96	59,00	43,54	30,56			38,98	27,92	24,48			43,19	30,28	25,59			36,53	19,44	8,54		

FIGURE 5.7: **Runtime analysis on NVIDIA Jetson TX2.** We report input resolution and FPS when varying the output scale.



		Nano (Max-N)					Nano (5W)				
Model	Image Res.	E	Q	H	F	Ref.	E	Q	H	F	Ref.
MonoResMatch	1280x384	1,17	1,13	0,98	-	-	0,96	0,91	0,77	-	-
MonoResMatch-RT	640x192	2,10	1,99	1,97	0,19	0,12	1,89	1,81	1,73	0.14	0.10
DepthHints	1024x320	0,95	0,91	0,85	0,76		0,75	0,72	0,65	0,59	
DepthHints-RT	640x192	1,66	1,63	1,55	1,45		1,35	1,28	1,24	1,17	
MonoDepth	512x256	2,06	1,99	1,92	1,75		1,66	1,59	1,50	1,35	
MonoDepth-RT	256x128	3,37	3,36	3,29	3,16		3,07	2,94	2,89	2,72	
3Net	512x256	1,57	1,50	1,38	1,20		1,29	1,21	1,12	0,97	
3Net-RT	256x128	2,36	2,27	2,24	2,08		2,16	2,03	1,94	1,82	
MonoDepth2	640x192	3,24	3,20	3,06	2,46		2,83	2,95	2,84	2,37	
MonoDepth2-RT	320x96	8,86	7,04	5,64	5,34		5,59	5,58	5,02	4,93	
FastDepth	512x256				3,22					2,84	
FastDepth-RT	256x128				4,42					3,69	
FastDepth	640x192				3,35					2,92	
FastDepth-RT	320x96				4,68					4,13	
PyD-Net	256x128	17,37	16,50	5,80			9,85	7,02	5,49		
PyD-Net++	640x192	17,52	16,66	5,59			10,00	7,74	5,59		
PyD-Net2	640x192	33,94	24,80	6,16			21,40	11,17	8,14		
PyD-Net2-RT	320x96	38,38	28,82	12,27			24,07	12,17	10,93		



		Nano Cortex A57 (Max-N)					Nano Cortex A57 (5W)				
Model	Image Res.	E	Q	H	F	Ref.	E	Q	H	F	Ref.
MonoResMatch	1280x384	0,27	0,23	0,19	-	-	0,10	0,08	0,07	-	-
MonoResMatch-RT	640x192	1,00	0,86	0,73	0,46	0,22	0,36	0,31	0,26	0,08	0,03
DepthHints	1024x320	0,20	0,19	0,16	0,15		0,07	0,07	0,06	0,05	
DepthHints-RT	640x192	0,51	0,47	0,42	0,38		0,18	0,17	0,15	0,14	
MonoDepth	512x256	0,70	0,63	0,55	0,48		0,26	0,23	0,20	0,17	
MonoDepth-RT	256x128	2,34	2,11	1,89	1,68		0,93	0,83	0,73	0,64	
3Net	512x256	0,58	0,49	0,40	0,33		0,21	0,17	0,14	0,12	
3Net-RT	256x128	1,99	1,71	1,43	1,21		0,77	0,65	0,54	0,45	
MonoDepth2	640x192	1,43	1,28	1,05	0,93		0,51	0,45	0,37	0,33	
MonoDepth2-RT	320x96	4,73	4,14	3,41	3,02		1,82	1,63	1,38	1,20	
FastDepth	512x256				1,90					0,76	
FastDepth-RT	256x128				6,04					2,83	
FastDepth	640x192				2,07					0,81	
FastDepth-RT	320x96				6,50					2,64	
PyD-Net	256x128	7,95	4,30	1,73			3,25	1,70	0,61		
PyD-Net++	640x192	8,48	4,58	1,85			3,30	1,80	0,67		
PyD-Net2	640x192	9,57	5,13	1,92			3,76	1,91	0,69		
PyD-Net2-RT	320x96	29,23	16,00	6,85			13,77	6,63	2,56		

FIGURE 5.8: **Runtime analysis on NVIDIA Jetson Nano.** We report input resolution and FPS when varying the output scale.

5.3.2 Runtime analysis on different architectures

Although most solutions in literature do not examine runtime requirements on devices different from high-end GPUs, many scenarios (e.g., those targeting the consumer and automotive market) rely on much less powerful devices often with severe constraints regarding power consumption and memory. Therefore, in this section, the time required to obtain a depth map on a large pool of heterogeneous devices characterized by different computing power and consumption is extensively studied. The following devices are deployed in our analysis: a high-end GPU (NVIDIA Titan Xp, $\sim$ 250W consumption), an average CPU system (Intel i7-6700k @4.2 Ghz, $\sim$ 90W consumption) an NVIDIA Jetson TX2 board, equipped with a GP10B GPU and 2 CPUs (a dual-core Denver 2 CPU and a quad-core ARM Cortex A57 2035Mhz, this latter used in our experiments), and an NVIDIA Jetson Nano, equipped with a 128-core NVIDIA Maxwell GPU and an ARM Cortex A57 CPU 1476Mhz. The outcome of the two benchmarks are shown, respectively, in Figure 5.6 (Titan Xp and i7), Figure 5.7 (Jetson TX2) and 5.8, reporting frames per second (FPS) achieved at each scale by the competitors in Table 5.2, with the addition of FastDepth [326] (without pruning and optimizations for a fair comparison) with both 512×256 and 640×192 inputs, and all PyD-Net configurations included in the ablation experiments. Forward time is measured excluding any overhead due to I/O, using Tensorflow profiling tools.

NVIDIA Titan Xp. Although state-of-the-art solutions run at reasonble speed [321, 302] and in some cases are extremely fast [80] on this high-end GPU, the gap with PyD-Net seen in Figure 5.6 (left) remains non-negligible. Observing the performance of all networks at the same, given scale H, Q or E, PyD-Net achieves about 1.5$\times$ speed-up compared to the fastest competitor MonoDepth2 [80] and up to 7$\times$ compared to the slowest DepthHints [321]. FastDepth runs faster than MonoDepth and MonoDepth2 when running at the same resolution, although the latter can exploit early-stop to achieve faster inference. PyD-Net++ yields slight improvements on this device compared to PyD-Net due to the different input scales. At the same time, PyD-Net2 runs 1.4$\times$ faster at H and 1.7$\times$ at E than PyD-Net, proving that the changes in architecture lead to benefits not only regarding accuracy but also for runtime. Finally, PyD-Net2-RT reaches the highest speed, running at about 370 FPS at H and more than 550 at E. Deploying half resolution images for the competitors yields slight performance improvement, although all RT models running at F result faster than their original H counterparts. However, this is not enough to compete with the speed achieved by our PyD-Net variants, which confirm to be the fastest architectures. Moreover, as highlighted by the results shown previously in Figure 5.5, the RT strategy introduces significant accuracy drops compared to early-stopping, which is also more flexible (not requiring to retrain networks).

i7-6700k. On this device, the need for compact architectures like ours is more evident, as visible from Figure 5.6 (right). Indeed, all state-of-the-art networks rarely reach 5 FPS at F. Only FastDepth [326] surpasses this barrier. Of course, RT variants are faster yet reaching up to about 15 FPS. As on GPU, RT variants of the competitors result faster than applying early-stopping, although they require a new training from scratch.

Conversely, PyD-Net and its variants can process nearly 10 FPS at H. Consistent speedups are obtained moving from PyD-Net to PyD-Net++ and PyD-Net2. Finally, PyD-Net2-RT runs extremely fast, about 50 FPS at H and 230 FPS at E scales. It is worth highlighting how the PyD-Net2-RT speedup is dramatic in this case, about 5$\times$ and 3$\times$ faster than PyD-Net2 at H and E, respectively. The following benchmarks will show how this behavior consistently occurs when using CPUs.

NVIDIA Jetson TX2. This board provides different configurations, enabling to trade performance for energy consumption. Concerning GPU, 3 main configurations exist: *Max-N* for maximum performance, *Max-Q* for minimum energy consumption, *Max-P Core-All* to enable both CPUs and average GPU performance (consuming less than 15 W). About CPU, *Max-P-ARM* configuration is considered, enabling maximum performance for ARM cores. Figure 5.7 reports a benchmark ran over the board, showing performance on both GPU (left) and CPU (right). Interestingly, according to Figures 5.6 and 5.7, in both *Max-Q* and *Max-N* modes Jetson TX2 grants performance often comparable to the i7-6700k CPU with much lower energy consumption (i.e., about 15 W vs. 90 W), in particular achieving faster inference for complex models such as MonoResMatch, DepthHints and their RT variants, except when stopping at E and Q, at which the i7 CPU shows larger speed-up whereas the TX2 GPU saturates.

Mid-range architectures such as MonoDepth and 3Net barely surpass the 5 FPS barrier on GPU while failing at reaching 1 FPS on CPU setups even at the lowest scale, similar to MonoResMatch DepthHints (rightmost table in the figure). Their RT variants are faster, despite never reaching 10 FPS, even on GPU. Faster models such as FastDepth and MonoDepth2 consistently run at around 10 FPS at F scale (8 in power-saving mode Max-Q), with the latter nearly reaching 20 FPS at E in Max-N mode and their RT variants processing up to 25 FPS. On the ARM processor, they run at about 2 FPS, with RT variants reaching 6-7 FPS.

Focusing on our architectures, all of them easily break the 15 FPS barrier at the highest scale on GPU, except for PyD-Net and PyD-Net++, reaching 13 FPS in power-save mode Max-Q. PyD-Net2 runs at more than 20 FPS (16 in Max-N) and about 40 FPS at E, with PyD-Net2-RT always reaching about 25 FPS at H and breaking the 35 FPS barrier at E (up to nearly 60 FPS in Max-N mode). With the CPU, all PyD-Net variants still break the 2 FPS barrier at H, with PyD-Net2-RT reaching 8.5 FPS. By stopping at the E scale, all variants reach 10 FPS. As with the i7 processor, PyD-Net2-RT achieves a massive speed-up at E, reaching 35 FPS and being about 5× faster than the most efficient competitor (FastDepth on 320 × 96 images).

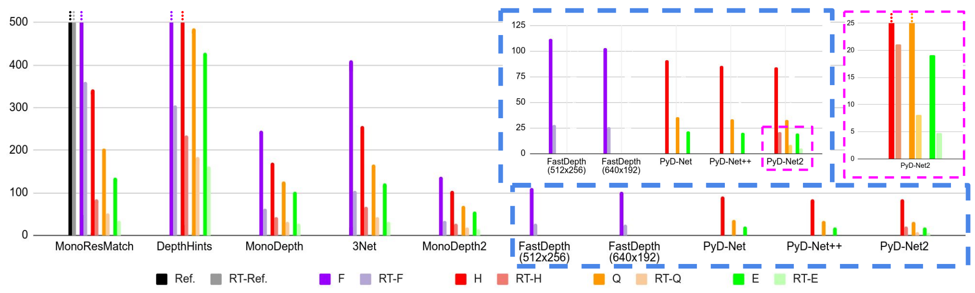


FIGURE 5.9: **Memory footprint (MB).** We report RAM usage for all models, testing on i7 CPU. We zoom over compact networks (blue) and further over PyD-Net2(purple). The dotted line means that memory usage is higher than the maximum value reported on the y-axis.

NVIDIA Jetson Nano. To conclude our benchmark, studies on the NVIDIA Jetson Nano board are shown in Figure 5.8, collecting measurements on the Maxwell GPU (top) and the Cortex A57 processor (bottom), both considering two power modes, *Max-N* and *5W*, respectively for maximum performance and minimum power consumption. Starting with most complex models, MonoResMatch cannot run at Ref. and F because of memory constraints. It and DepthHints barely reach 1FPS on the GPU, always requiring not less than 5 seconds per frame on CPU. Their RT variants barely reach 2 and 1 FPS on GPU and CPU, respectively.

Moving to the remaining models, none of them reaches 5 FPS on GPU and 10 FPS with RT variants, while barely reaching 2 FPS on the ARM processor, getting closer to 5 with their RT variants (except for FastDepth-RT running at about 6 FPS). On the contrary, all PyD-Net variants on GPU always break the 5 FPS barrier, with PyD-Net2 and its RT variant processing more than 30 and 20 FPS at E in Max-N and 5W modes. On the Cortex A57 CPU, all variants get close to the 2 FPS barrier on Max-N mode, although not reaching 1 FPS yet in 5W mode. However, early-stopping allows most PyD-Net variants to achieve 8-9 FPS and 3 FPS in the two modalities, with PyD-Net2-RT again achieving a massive speed-up at E and processing almost 30 and 14 FPS in Max-N and 5W.

5.3.3 Memory footprint

Finally, the memory footprint concerning inference of a single depth map is measured. Figure 5.9 plots the memory consumption[2] for each of the networks studied so far and their RT variants. The amount of memory (in MB) required by the network to infer the result is reported, cutting off any overhead due to input/output and other tasks not concerned with depth estimation, taken into account in our previous work [230]. MonoResMatch and DepthHints requires more than 500 MB for their highest scales, but they behave differently at lower scales with the former falling below 150 MB and the latter being still above 400. Thanks to the lower input resolution, RT variants require reduced memory. However, MonoResMatch-RT still requires more than 500MB to run a full inference, dropping to about 50MB at the lowest scale, while DepthHints-RT requires 300 to 150 MB.

MonoDepth drops below 250MB at F, yet requiring more than 100 at E, while 3Net requires much more memory because of the double decoder. MonoDepth2 and FastDepth have similar requirements at F, with MonoDepth2 being able to fall to nearly 50MB thanks to early-stopping at E. RT variants drop memory requirements below 100 MB at F, getting below 50 MB in most cases at Q and E. FastDepth, both at 512×256 and 640×192 requires about 100 MB, and 25 MB when processing half resolution images in input.

The original PyD-Net [230], even at H, requires less than the minimum memory used by most of the models except MonoDepth2, dropping to 21 MB at E. By changing the input resolution and the network architecture additional savings on memory are achieved, emphasizing that PyD-Net2 is not only more accurate and faster than PyD-Net but also efficient regarding memory requirements. Finally, PyD-Net2-RT has a very tiny memory footprint compared to the other models. Indeed, at H scale, it requires about the same memory required by other PyD-Net variants at E, while running at such scale requires less than 5 MB. Thus, thanks to its shallow memory footprint, the high frame rate even on the Denver CPU, and the acceptable accuracy, PyD-Net2-RT is a perfect candidate to quickly infer depth from monocular images on embedded and consumer devices.

5.4 Conclusions

In this chapter we have introduced the efficient architectures of the PyD-Net family for monocular depth estimation. As competitor methods [302, 321], it can be trained

[2]measured using Tensorboard tools.

in weakly-supervised manner on rectified stereo pairs to achieve a fairly good accuracy. However, the peculiar design makes these networks suited for real-time applications on standard CPUs and also enables its effective deployment on embedded systems. Although the baseline model PyD-Net showed promising results concerning accuracy and performance, the novel PyD-Net2 improves both aspects, reducing at the same time the gap to huge state-of-the-art architectures. Moreover, PyD-Net2-RT achieves reasonably accurate depth map further pushing ahead performance especially when targeting low power devices, reaching more than 10 FPS even on a low-power Denver CPU in its fastest configuration, paving the way to an effective deployment of real-time deep learning based depth estimation to a broader set of devices.

Chapter 6

Comprehensive scene understanding from videos

In the previous chapters, we have dealt with monocular depth models. However, as we already pointed out in Chapters 1 and 2, geometric information about the scene is paramount in many applications, but may not be enough to completely solve all the problems. For instance, in the context of self-driving cars, depth is crucial to avoid obstacles, but it does not provide any cues about the *type* of the obstacle. At the same time, avoiding a pedestrian is much more important than dodging a piece of wood. Past works in the literature deployed a single model that can be specialized on each task, or a mixture of models, i.e. a set of task-specialized yet independent networks that are optimized together. The work of Eigen et al. [64] belongs to the first category, because the same model has been used for depth, semantic and surface normal estimation tasks. The authors adopted a supervised training strategy for all the tasks, which is expensive for practical applications. In the second family, we find the work of Ma et al. [182], in which large networks have been employed for each task, at the cost of high computational costs.

Conversely, this chapter presents a single framework for comprehensive scene understanding from monocular videos able to exploit the relationships between different tasks. The multi-stage network architecture, named ΩNet, can predict depth, semantics, optical flow, per-pixel motion probabilities and motion masks given a monocular video as input. This comes alongside with estimating the pose between adjacent frames for an uncalibrated camera, whose intrinsic parameters are also estimated. The training methodology leverages on self-supervision, knowledge distillation and multi-task learning. Moreover, ΩNet is lightweight, counting less than 8.5 million parameters, and fast, as it can run at nearly 60 FPS and 5 FPS on an NVIDIA Titan Xp and a Jetson TX2, respectively. The material of this chapter has been published as Distilled Semantics for Comprehensive Scene Understanding from Videos in [9].

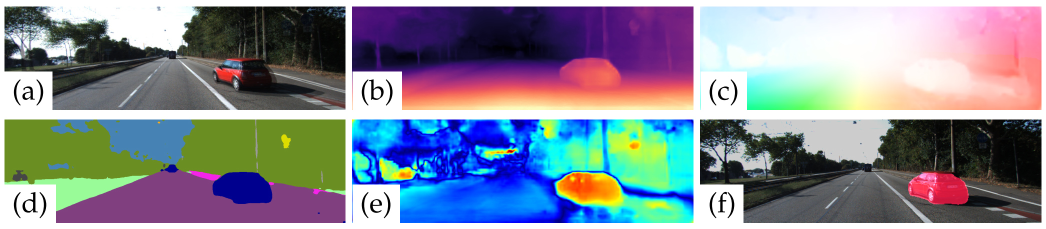


FIGURE 6.1: **Outputs of ΩNet framework.** Given an input monocular video (a), our network can provide the following outputs in real-time: depth (b), optical flow (c), semantic labels (d), per-pixel motion probabilities (e), motion mask (f).

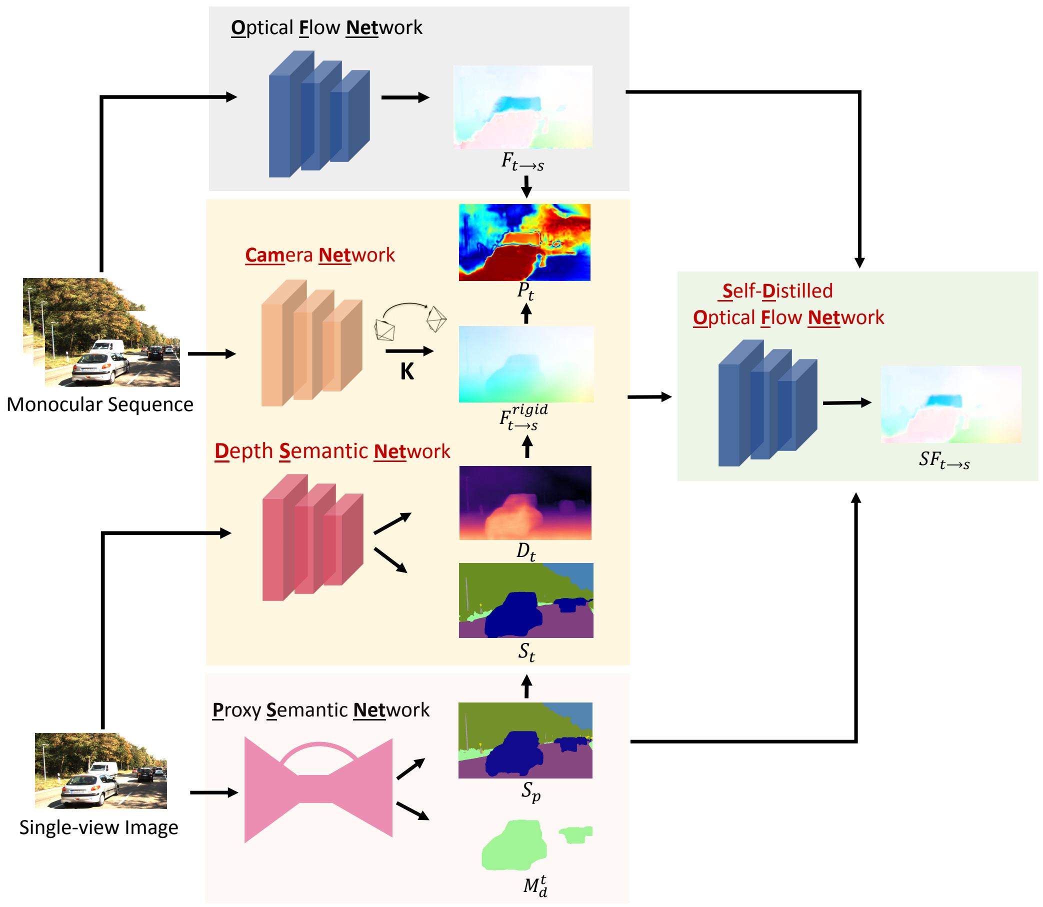


FIGURE 6.2: **ΩNet framework.** Overall framework for training ΩNet to predict depth, camera pose, camera intrinsics, semantic labels and optical flow. In **red** architectures composing ΩNet.

6.1 Overall learning framework

Our goal is to develop a real-time comprehensive scene understanding framework capable of learning strictly related tasks from monocular videos. Purposely, we propose a multi-stage approach to learn first geometry and semantics, then elicit motion information, as depicted in Figure 6.2.

6.2 Geometry and semantics

6.2.1 Self-supervised depth and pose estimation.

We propose to solve a self-supervised single-image depth and pose estimation problem by exploiting geometrical constraints in a sequence of N images, in which one of the frames is used as the target view I_t and the other ones in turn as the source image I_s. Assuming a moving camera in a stationary scene, given a depth map D_t aligned with I_t, the camera intrinsic parameters K and the relative pose $T_{t \to s}$ between I_t and I_s, it is possible to sample pixels from I_s in order to synthesise a warped image $\tilde{I}_t$ aligned with I_t. The mapping between corresponding homogeneous pixels coordinates $p_t \in I_t$ and $p_s \in I_s$ is given by:

$$p_s \sim K T_{t \to s} D_t K^{-1} p_t \tag{6.1}$$

Following [365], we use the sub-differentiable bilinear sampler mechanism proposed in [122] to obtain $\tilde{I}_t$. Thus, in order to learn depth, pose and camera intrinsics we train two separate CNNs to minimize the photometric reconstruction error between $\tilde{I}_t$ and I_t, defined as:

$$\mathcal{L}_{ap}^{D} = \sum_{p} \psi(I_t(p), \tilde{I}_t(p)) \tag{6.2}$$

where ψ is a photometric error function between the two images. However, as pointed out in [80], such a formulation is prone to errors at occlusion/disocclusion regions or in static camera scenarios. To soften these issues, we follow the same principles as suggested in [80], where a minimum per-pixel reprojection loss is used to compute the photometric error, an automask method allows for filtering-out spurious gradients when the static camera assumption is violated, and an edge-aware smoothness loss term is used as in [79]. Moreover, we use the depth normalization strategy proposed in [314]. We compute the rigid flow between I_t and I_s as the difference between the projected and original pixel coordinates in the target image:

$$F_{t \to s}^{rigid}(p_t) = p_s - p_t \tag{6.3}$$

6.2.2 Distilling semantic knowledge.

The proposed distillation scheme is motivated by how time-consuming and cumbersome obtaining accurate pixel-wise semantic annotations is. Thus, we train our framework to estimate semantic segmentation masks S_t by means of supervision from cheap proxy labels S_p distilled by a semantic segmentation network called Proxy Semantic Network, which has been pre-trained on few annotated samples and is capable to generalize well to diverse datasets. Availability of proxy semantic labels for the frames of a monocular video enables us to train a single network to predict jointly depth and semantic labels. Accordingly, the joint loss is obtained by adding a standard cross-entropy term $\mathcal{L}_{sem}$ to the previously defined self-supervised image reconstruction loss $\mathcal{L}_{ap}^{D}$. Moreover, similarly to [352], we deploy a cross-task loss term, $\mathcal{L}_{edge}^{D}$, aimed at favouring spatial coherence between depth edges and semantic boundaries. However, unlike [352], we do not exploit stereo pairs at training time.

6.2.3 Optical flow and motion segmentation

Self-supervised optical flow. As the 3D structure of a scene includes stationary as well as non-stationary objects, to handle the latter we rely on a classical optical flow formulation. Formally, given two images I_t and I_s, the goal is to estimate the 2D motion vectors $F_{t \to s}(p_t)$ that map each pixel in I_t into its corresponding one in I_s. To learn such a mapping without supervision, previous approaches [194, 170, 347] employ an image reconstruction loss $\mathcal{L}_{ap}^{F}$ that minimizes the photometric differences between I_t and the back-warped image $\tilde{I}_t$ obtained by sampling pixels from I_s using the estimated 2D optical flow $F_{t \to s}(p_t)$. This approach performs well for non-occluded pixels but provides misleading information within occluded regions.

Pixel-wise motion probability. Non-stationary objects produce systematic errors when optimizing $\mathcal{L}_{ap}^{D}$ due to the assumption that the camera is the only moving body in an otherwise stationary scene. However, such systematic errors can be exploited to identify non-stationary objects: at pixels belonging to such objects the rigid flow $F^{rigid}_{t\to s}$ and the optical flow $F_{t\to s}$ should exhibit different directions and/or norms. Therefore, a pixel-wise probability of belonging to an object independently moving between frames s and t, P_t, can be obtained by normalizing the differences between the two vectors. Formally, denoting with $\theta(p_t)$ the angle between the two vectors at location p_t, we define the per-pixel motion probabilities as:

$$P_t(p_t) = \max\{\frac{1-\cos\theta(p_t)}{2}, 1-\eta(p_t)\} \tag{6.4}$$

where $\cos\theta(p_t)$ can be computed as the normalized dot product between the vectors and evaluates the similarity in direction between them, while $\eta(p_t)$ is defined as

$$\eta(p_t) = \frac{\min\{\|F_{t\to s}(p_t)\|_2, \|F^{rigid}_{t\to s}(p_t)\|_2\}}{\max\{\|F_{t\to s}(p_t)\|_2, \|F^{rigid}_{t\to s}(p_t)\|_2\}}, \tag{6.5}$$

i.e. a normalized score of the similarity between the two norms. By taking the maximum of the two normalized differences, we can detect moving objects even when either the directions or the norms of the vectors are similar. A visualization of $P_t(p_t)$ is depicted in Figure 6.3(d).

Semantic-aware self-distillation paradigm. Finally, we combine semantic information, estimated optical flow, rigid flow and pixel-wise motion probabilities within a final training stage to obtain a more robust self-distilled optical flow network. In other words, we train a new instance of the model to infer a self-distilled flow $\text{SF}_{t\to s}$ given the estimates $\text{F}_{t\to s}$ from a first self-supervised network and the aforementioned cues. As previously discussed and highlighted in Figure 6.3(c), standard self-supervised optical flow is prone to errors in occluded regions due to the lack of photometric information but can provide good estimates for the dynamic objects in the scene. On the contrary, the estimated rigid flow can properly handle occluded areas thanks to the minimum-reprojection mechanism [80]. Starting from these considerations, our key idea is to split the scene into stationary and potentially dynamics objects, and apply on them the proper supervision. Purposely, we can leverage several observations:

1. **Semantic priors.** Given a semantic map S_t for image I_t, we can binarize pixels into static M^s_t and potentially dynamic M^d_t, with $M^s_t \cap M^d_t = \emptyset$. For example, we expect that points labeled as *road* are static in the 3D world, while pixels belonging to the semantic class *car* may move. In M^d_t, we assign 1 for each potentially dynamic pixel, 0 otherwise, as shown in Figure 6.3(e).

2. **Camera motion boundary mask.** Instead of using a backward-forward strategy [367] to detect boundaries occluded due to the ego-motion, we analytically compute a binary boundary mask M^b_t from depth and ego-motion estimates as proposed in [187]. We assign a 0 value for out-of-camera pixels, 1 otherwise as shown in Figure 6.3(f).

3. **Consistency mask.** Because the inconsistencies between the rigid flow and $\text{F}_{t\to s}$ are not only due to dynamic objects but also to occluded/inconsistent areas, we can leverage Equation (6.4) to detect such critical regions. Indeed, we define the consistency mask as:

$$M^c_t = P_t < \xi, \xi \in [0,1] \tag{6.6}$$

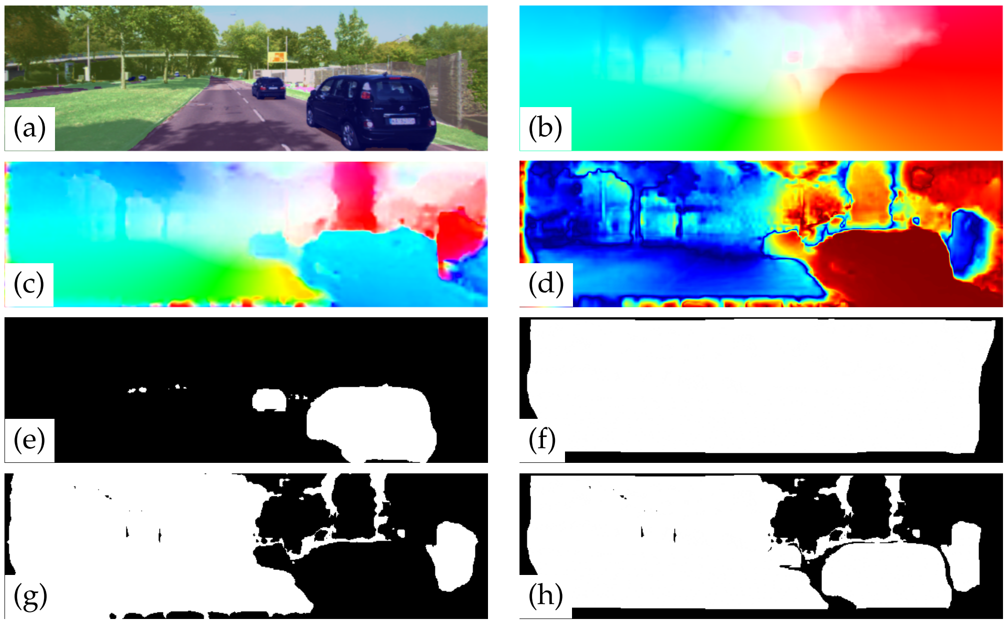


FIGURE 6.3: **Overview of the semantic-aware and self-distilled optical flow estimation approach.** We leverage semantic segmentation S_t (a) together with rigid flow $F^{rigid}_{t\to s}$ (b), teacher flow $F_{t\to s}$ (c) and motion probabilities P_t (d), the warmer the higher. From a) we obtain semantic priors M^d_t (e), combined with boundary mask M^b_t (f) and consistency mask M^c_t (g) derived from (d) as in Equation 6.6, in order to obtain the final mask M (h) as in Equation 6.7.

This mask assigns 1 where the condition is satisfied, 0 otherwise (i.e. inconsistent regions) as in Figure 6.3(g).

Finally, we compute the final mask M, in Figure 6.3(h), as:

$$M = \min\{\max\{M^d_t, M^c_t\}, M^b_t\} \tag{6.7}$$

As a consequence, M will effectively distinguish regions in the image for which we can not trust the supervision sourced by $F_{t\to s}$, i.e. inconsistent or occluded areas. On such regions, we can leverage our proposed self-distillation mechanism. Then, we define the final total loss for the self-distilled optical flow network as:

$$\mathcal{L} = \sum \alpha_r \phi(SF_{t\to s}, F^{rigid}_{t\to s}) \cdot (1 - M) + \alpha_d \phi(SF_{t\to s}, F_{t\to s}) \cdot M + \psi(I_t, \tilde{I}^{SF}_t) \cdot M \tag{6.8}$$

where ϕ is a distance function between two motion vectors, while α_r and α_d are two hyper-parameters.

6.2.4 Motion segmentation

At test time, from pixel-wise probability P_t computed between $SF_{t\to s}$ and $F^{rigid}_{t\to s}$, semantic prior M^d_t and a threshold τ, we compute a motion segmentation mask by:

$$M^{mot}_t = M^d_t \cdot (P_t > \tau), \tau \in [0, 1] \tag{6.9}$$

Such mask allows us to detect moving objects in the scene independently of the camera motion. A qualitative example is depicted in Figure 6.1(f).

6.3 Experimental results

Using standard benchmark datasets, we present here the experimental validation on the main tasks tackled by ΩNet. As depicted in Figure 6.2, our frameworks contains a single lightweight module DSNet for depth and semantic estimation, and another network CamNet that predicts camera intrinsic and extrinsic parameters. Two identical models OFNet and SD-OFNet generate optical flow maps. OFNet is trained using photometric supervision and predicts $F_{t\to s}$, while the self-distilled network SD-OFNet is trained according to the loss term defined in Equation 6.8 and predicts $SF_{t\to s}$. It is worth noticing that, at test time, neither the Proxy Semantic Network nor OFNet are used because their labels are used for training purposes only. We set $N = 3$, i.e. we adopt 3-frames video sequences. Appendix C provides additional details regarding these architectures and how to train them.

6.3.1 Monocular depth estimation

In this section, we compare our results to other state-of-the-art proposals and assess the contribution of each component to the quality of our monocular depth predictions.

Comparison with state-of-the-art. We compare with state-of-the-art self-supervised networks trained on monocular videos using the Eigen split of KITTI [65] (K). Since the predicted depth is defined up to a scale factor, we align the scale of our estimates by multiplying them by a scalar that matches the median of the ground-truth, as in [365]. Table 6.1 reports extensive comparison with respect to several monocular depth estimation methods. We outperform our main competitors such as [347, 367, 52, 15] that solve multi-task learning or other strategies that exploit additional information during the training/testing phase [40, 335]. Moreover, our best configuration, i.e. pre-training on Cityscapes [55] (CS) and using 1024×320 resolution, achieves better results in 5 out of 7 metrics with respect to the single-task, state-of-the-art proposal [80] (and is the second best and very close to it on the remaining 2) which, however, leverages on a larger ImageNet pre-trained model based on ResNet18. It is also interesting to note how our proposal without pretraining obtains the best performance in 6 out of 7 measures on 640×192 images (row 1 vs 15). These results validate our intuition about how the use of semantic information can guide geometric reasoning and make a compact network provide state-of-the-art performance even with respect to larger and highly specialized depth-from-mono methods. Finally, we also report the results achieved by the methods illustrated in Chapters 4 and 5: despite both MonoResMatch and PyD-Net2 exploit stereo pairs and not monocular videos at training time, the model performs better than the lightweight PyD-Net2 and shows a limited gap with the much larger model MonoResMatch.

Ablation study. Table 6.2 highlights how progressively adding the key innovations proposed in [82, 80, 314] contributes to strengthen ΩNet, already comparable to other methodologies even in its baseline configuration (first row). Interestingly, a large improvement is achieved by deploying joint depth and semantic learning (rows 5 vs 7), which forces the network to simultaneously reason about geometry and content within the same shared features. By replacing DSNet within ΩNet

Method	M	A	I	CS	Abs Rel	Sq Rel	RMSE	log RMSE	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Godard et al.[80]					0.132	1.044	5.142	0.210	0.845	0.948	0.977
Godard et al.[80] (1024 × 320)			✓		**0.115**	0.882	**4.701**	**0.190**	**0.879**	**0.961**	**0.982**
Zhou et al.[364]			✓		0.121	**0.837**	4.945	0.197	0.853	0.955	**0.982**
Mahjourian et al.[187]				✓	0.159	1.231	5.912	0.243	0.784	0.923	0.970
Wang et al.[314]				✓	0.151	1.257	5.583	0.228	0.810	0.936	0.974
Bian et al.[25]				✓	0.128	1.047	5.234	0.208	0.846	0.947	0.970
Yin et al.[347]	✓			✓	0.153	1.328	5.737	0.232	0.802	0.934	0.972
Zou et al.[367]	✓			✓	0.146	1.182	5.215	0.213	0.818	0.943	0.978
Chen et al.[52]	✓		✓		0.135	1.070	5.230	0.210	0.841	0.948	0.980
Luo et al.[178]	✓				0.141	1.029	5.350	0.216	0.816	0.941	0.976
Ranjan et al.[15]	✓				0.139	1.032	5.199	0.213	0.827	0.943	0.977
Xu et al.[335]		✓	✓		0.138	1.016	5.352	0.217	0.823	0.943	0.976
Casser et al.[40]		✓			0.141	1.026	5.290	0.215	0.816	0.945	0.979
Gordon et al.[82]	✓	✓			0.128	0.959	5.230	-	-	-	-
MonoResMatch		✓			0.111	0.867	4.714	0.199	0.864	0.954	0.979
MonoResMatch		✓		✓	**0.096**	**0.673**	**4.351**	**0.184**	**0.890**	**0.961**	**0.981**
PyD-Net2		✓		✓	0.127	1.059	5.259	0.218	0.834	0.942	0.974
PyD-Net2-RT		✓		✓	0.145	1.260	5.773	0.236	0.797	0.925	0.970
ΩNet(640 × 192)	✓	✓			0.126	0.835	4.937	0.199	0.844	0.953	0.982
ΩNet(1024 × 320)	✓	✓			0.125	0.805	4.795	0.195	0.849	0.955	0.983
ΩNet(640 × 192)	✓	✓		✓	0.120	0.792	4.750	0.191	0.856	0.958	0.984
ΩNet(1024 × 320)	✓	✓		✓	**0.118**	**0.748**	**4.608**	**0.186**	**0.865**	**0.961**	**0.985**

TABLE 6.1: **Depth evaluation on the Eigen split [65] of KITTI [196].** We indicate additional features of each method. M: multi-task learning, A: additional information (e.g. object knowledge, semantic information), I: feature extractors pre-trained on ImageNet [58], CS: network pre-trained on Cityscapes [55]

with a larger backbone [347] (rows 5 vs 6) we obtain worse performance, validating the design decisions behind our compact model. Finally, by pre-training on CS we achieve the best accuracy, which increases alongside with the input resolution (rows 8 to 10).

Resolution	Learned Intr. [82]	Norm. [314]	Min. Repr. [80]	Automask [80]	Sem. [48]	Pre-train	Abs Rel	Sq Rel	RMSE	log RMSE	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
640 × 192	-	-	-	-	-	-	0.139	1.056	5.288	0.215	0.826	0.942	0.976
640 × 192	✓	-	-	-	-	-	0.138	1.014	5.213	0.213	0.829	0.943	0.977
640 × 192	✓	✓	-	-	-	-	0.136	1.008	5.204	0.212	0.832	0.944	0.976
640 × 192	✓	✓	✓	-	-	-	0.132	0.960	5.104	0.206	0.840	0.949	0.979
640 × 192	✓	✓	✓	✓	-	-	0.130	0.909	5.022	0.207	0.842	0.948	0.979
640 × 192 †	✓	✓	✓	✓	-	-	0.134	1.074	5.451	0.213	0.834	0.946	0.977
640 × 192	✓	✓	✓	✓	✓	-	0.126	0.835	4.937	0.199	0.844	0.953	0.980
416 × 128	✓	✓	✓	✓	✓	✓	0.126	0.862	4.963	0.199	0.846	0.952	0.981
640 × 192	✓	✓	✓	✓	✓	✓	0.120	0.792	4.750	0.191	0.856	0.958	0.984
1024 × 320	✓	✓	✓	✓	✓	✓	0.118	0.748	4.608	0.186	0.865	0.961	0.985

TABLE 6.2: **Ablation study of depth network on the Eigen split [65] of KITTI.** †: our network is replaced by a ResNet50 backbone [347].

Depth Range Error Analysis. We dig into our depth evaluation to explain the effectiveness of ΩNet with respect to much larger networks. Table 6.3 compares, at different depth ranges, our model with more complex ones [80, 347]. This experiment shows how ΩNet superior performance comes from better estimation of large depths: ΩNet outperforms both competitors when we include distances larger than 8 m in the evaluation, while it turns out less effective in the close range.

6.3.2 Semantic segmentation

In Table 6.4, we report the performance of ΩNet on semantic segmentation for the 19 evaluation classes of CS according to the metrics defined in [55, 17]. We compare ΩNet against state-of-the art networks for real-time semantic segmentation [45, 158] when training on CS and testing either on the validation set of CS (rows 1-3)

Method	Cap (m)	Abs Rel	Sq Rel	RMSE	RMSE log
Godard et al.[80]	0-8	**0.059**	**0.062**	0.503	**0.082**
ΩNet†	0-8	0.060	0.063	**0.502**	**0.082**
ΩNet	0-8	0.062	0.065	0.517	0.085
Godard et al.[80]	0-50	0.125	0.788	3.946	0.198
ΩNet†	0-50	0.127	0.762	4.020	0.199
ΩNet	0-50	**0.124**	**0.702**	**3.836**	**0.195**
Godard et al.[80]	0-80	0.132	1.044	5.142	0.210
ΩNet†	0-80	0.134	1.074	5.451	0.213
ΩNet	0-80	**0.126**	**0.835**	**4.937**	**0.199**

TABLE 6.3: **Depth errors by varying the range.** †: our network is replaced by a ResNet50 backbone [347].

Method	Train	Test	mIoU class	mIoU cat.	Pixel Acc.
DABNet [158]	CS(S)	CS	69.62	87.56	94.62
FCHardNet [45]	CS(S)	CS	**76.37**	**89.22**	**95.35**
ΩNet	CS(P)	CS	54.80	82.92	92.50
DABNet [158]	CS(S)	K	35.40	61.49	80.50
FCHardNet [45]	CS(S)	K	**44.74**	68.20	72.07
ΩNet	CS(P)	K	43.80	**74.31**	**88.31**
ΩNet	CS(P) + K(P)	K	46.68	75.84	88.12

TABLE 6.4: **Semantic segmentation on Cityscapes (CS) and KITTI (K)**. S: training on ground-truth, P: training on proxy labels. Results are expressed as percentages.

or the 200 semantically annotated images of K (rows 4-6). Even though our network is not as effective as the considered methods when training and testing on the same dataset, it shows greater generalization capabilities to unseen domains: it significantly outperforms other methods when testing on K for $\text{mIoU}_{\text{category}}$ and pixel accuracy, and provides similar results to [45] for $\text{mIoU}_{\text{class}}$. We relate this ability to our training protocol based on proxy labels (P) instead of ground-truths (S). Moreover, as we have already effectively distilled the knowledge from DPC [48] during pre-training on CS, there is only a slight benefit in training on both CS and K (with proxy labels only) and testing on K (row 7). Finally, although achieving 46.68 mIoU on fine segmentation, we obtain 89.64 mIoU for the task of segmenting static from potentially dynamic classes, an important result to obtain accurate motion masks.

6.3.3 Optical flow

In Table 6.5, we compare the performance of our optical flow network with competing methods using the KITTI 2015 stereo/flow training set [196] as testing set, which contains 200 ground-truth optical flow measurements for evaluation. We exploit all the raw K images for training, but we exclude the images used at testing time as done in [367] , to be consistent with experimental results of previous self-supervised optical flow strategies [347, 367, 52, 15]. From the table, we can observe how our self-distillation strategy allows SD-OFNet to outperform by a large margin

		train			test
Method	Dataset	EPE Noc	EPE All	Fl	Fl
Meisteret al.[194] - C	SYN + K	-	8.80	28.94	29.46
Meister et al.[194] - CSS	SYN + K	-	8.10	23.27	23.30
Zou et al.[367]	SYN + K	-	8.98	26.0	25.70
Ranjan et al.[15]	SYN + K	-	5.66	20.93	25.27
Wang et al.[318] **	K	-	5.58	-	18.00
Yin et al.[347]	K	8.05	10.81	-	-
Chen et al.[52] †	K	5.40	8.95	-	-
Chen et al.[52] (online) †	K	4.86	8.35	-	-
Ranjan et al.[15]	K	-	6.21	26.41	-
Luo et al.[178]	K	-	5.84	-	21.56
Luo et al.[178] *	K	-	5.43	-	20.61
ΩNet(Ego-motion)	K	11.72	13.50	51.22	-
OFNet	K	3.48	11.61	25.78	-
SD-OFNet	K	**3.29**	**5.39**	**20.0**	**19.47**

TABLE 6.5: **Optical flow evaluation on the KITTI 2015 dataset.** †: pre-trained on ImageNet, SYN: pre-trained on SYNTHIA [253], *: trained on stereo pairs, **: using stereo at testing time.

competitors trained on K only (rows 5-11), and it even performs better than models pre-initialized by training on synthetic datasets [253]. Moreover, we submitted our flow predictions to the online KITTI flow benchmark after retraining the network including images from the whole official training set. In this configuration, we can observe how our model achieves state-of-the-art *Fl* performances with respect to other monocular multi-task architectures.

6.3.4 Motion segmentation

In Table 6.6 we report experimental results for the motion segmentation task on the KITTI 2015 dataset, which provides 200 images manually annotated with motion labels for the evaluation. We compare our methodology with respect to other state-of-the-art strategies that performs multi-task learning and motion segmentation [15, 178, 318] using the metrics and evaluation protocol proposed in [178]. It can be noticed how our segmentation strategy outperforms all the other existing methodologies by a large margin. This demonstrates the effectiveness of our proposal to jointly combine semantic reasoning and motion probability to obtain much better results. We also report, as upper bound, the accuracy enabled by injecting semantic proxies [48] in place of ΩNet semantic predictions to highlight the low margin between the two.

6.3.5 Runtime analysis

Finally, we measure the runtime of ΩNet on different hardware devices, i.e. a Titan Xp GPU, an embedded NVIDIA Jetson TX2 board and an Intel i7-7700K@4.2 GHz CPU. Timings averaged over 200 frames at 640 $\times$ 192 resolution. Moreover, as each

Method	Pixel Acc.	Mean Acc.	mIoU	f.w. IoU
Yang et al.[343] *	0.89	0.75	0.52	0.87
Luo et al.[178]	0.88	0.63	0.50	0.86
Luo et al.[178] *	0.91	0.76	0.53	0.87
Wang et al.[318] (Full) **	0.90	0.82	0.56	0.88
Ranjan et al.[15]	0.87	0.79	0.53	0.85
ΩNet	**0.98**	**0.86**	**0.75**	**0.97**
ΩNet(Proxy [48])	0.98	0.87	0.77	0.97

TABLE 6.6: **Motion segmentation evaluation on the KITTI 2015 dataset.** * means trained on stereo pairs, while ** means using stereo at testing time.

component of ΩNet may be used on its own, we report the runtime for each independent task. As summarized in Table 6.7, our network runs in real-time on the Titan Xp GPU and at about 2.5 FPS on a standard CPU. It also fits the low-power NIVIDA Jetson TX2, achieving 4.5 FPS to compute all the outputs.

Device	Watt	D	DS	OF	Cam	Ω
Jetson TX2	15	12.5	10.3	6.5	49.2	4.5
i7-7700K	91	5.0	4.2	4.9	31.4	2.4
Titan XP	250	170.2	134.1	94.1	446.7	57.4

TABLE 6.7: **Runtime analysis on different devices.** We report the power consumption in Watt and the FPS. D: Depth, S: Semantic, OF: Optical Flow, Cam: camera pose, Ω: Overall architecture.

6.3.6 Additional qualitative examples

We report additional qualitative examples for motion segmentation, optical flow, depth and segmentation. Figure 6.5 we better visualize the difference between the teacher and student optical flow prediction for two samples from KITTI. We can notice how the flow from the student is much more smooth and less noisy than the teacher one. We also visualize in Figure 6.6 depth and semantic predictions of the model on Cityscapes dataset, while in Figure 6.7 we show two examples of motion segmentation masks.

6.4 Conclusions

This chapter presented ΩNet, a comprehensive scene understanding framework able to estimate the depth and the semantic label for each pixel given a single image, but also the optical flow between corresponding pixels when more images are given at test time. Moreover, the combination of semantic segmentation, optical flow and depth allows to detect accurately moving objects in the scene.

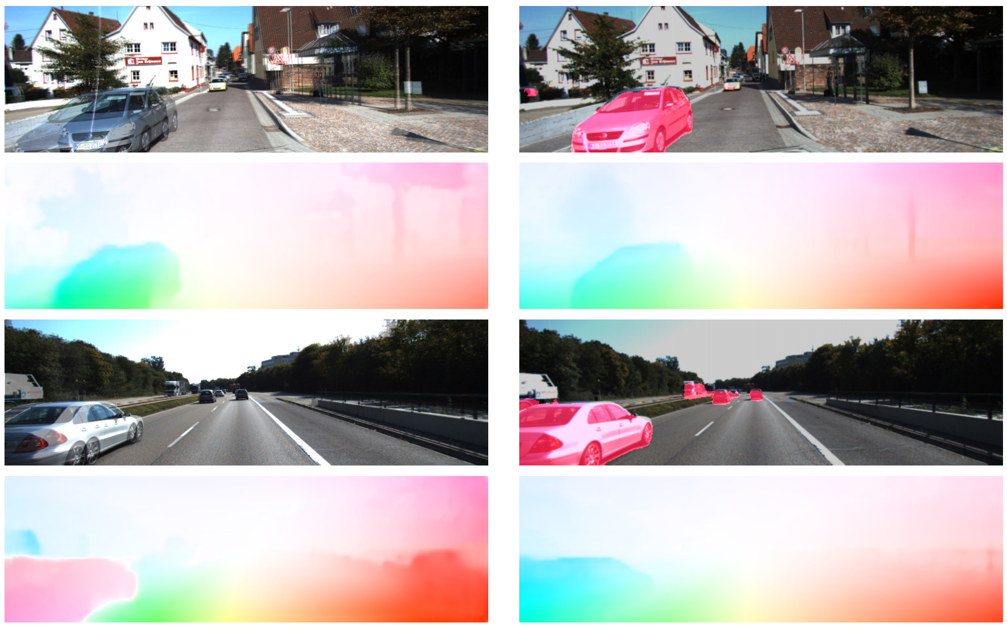


FIGURE 6.4: **Qualitative examples on motion segmentation and optical flow.** For each group of samples, first row depicts from left to right the input images and the moving objects detected in the scene by our method (highlighted in red). Second row illustrates the optical flow and the rigid flow respectively.

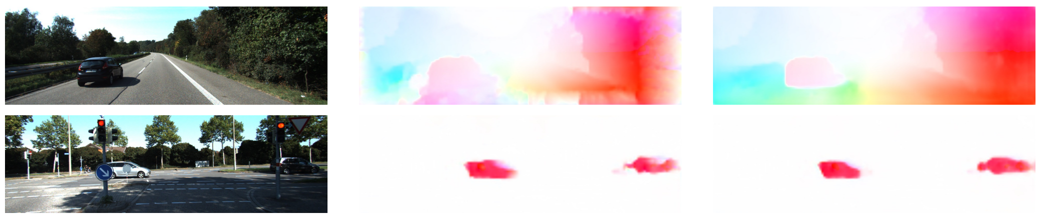


FIGURE 6.5: **Qualitative comparison between teacher and student flow networks.** From left to right, the input images, the teacher optical flow and the student prediction.

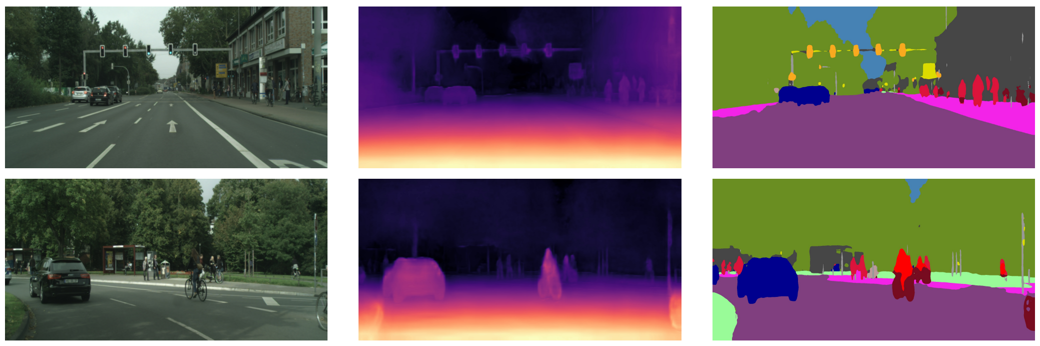


FIGURE 6.6: **Qualitative examples for depth and semantic on Cityscapes.** First column shows the reference image, while second and third columns depict the depth and semantic map predicted by ΩNet.

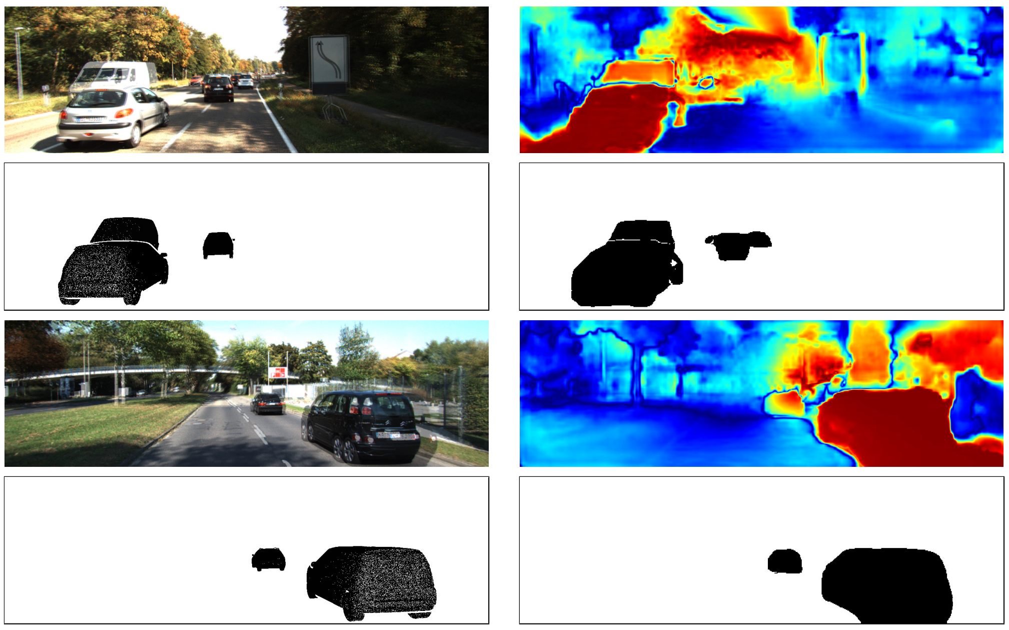


FIGURE 6.7: **Motion segmentation results.** From left to right, we show the monocular sequence, the outcome of the proposed motion probability strategy (high probability of motion is encoded in red), ground-truth motion masks and ours estimated motion segmentation masks.

Chapter 7

Monocular depth estimation in the wild

In Chapter 5 we introduced PyD-Net, a lightweight network for monocular depth estimation trained without the need for ground-truth labels. If compared to fully-supervised strategies, self-supervision (from videos or stereo pairs) is extremely convenient in terms of costs and requirements. Unfortunately, self-supervision does not provide a strong guidance in challenging regions, for instance in texture-less areas of difficult scenes. Moreover, both supervised and self-supervised learning based methods may suffer the domain gap issue, in particular when we test our models on environments different from the training one (e.g., training in indoor environments and testing on outdoor scenarios). Figure 7.1 illustrates the issue for the supervised model FastDepth [326] and the self-supervised MonoDepth2. The models have been trained indoor and outdoor respectively, and provide good qualitative maps for in-domain samples. However, when testing using out-domain images, i.e. outdoor for FastDepth and indoor for MonoDepth2, it is easy to notice that predictions are far from being perfect. As we already pointed out in Chapter 2, this is a well-known problem in literature and it has been faced by several works. MiDaS [245] by Ranftl et al. represents the state-of-the-art. The authors propose an effective yet time-consuming pipeline used to train a large model. Even if the issue related to model size can be easily addressed (the authors recently trained also a lightweight model), how to speed up the label-generation pipeline is an open problem. Our question is: can we *distill* the knowledge from these off-the-shelf models, trained in the wild, and *infuse* it to any arbitrary model?

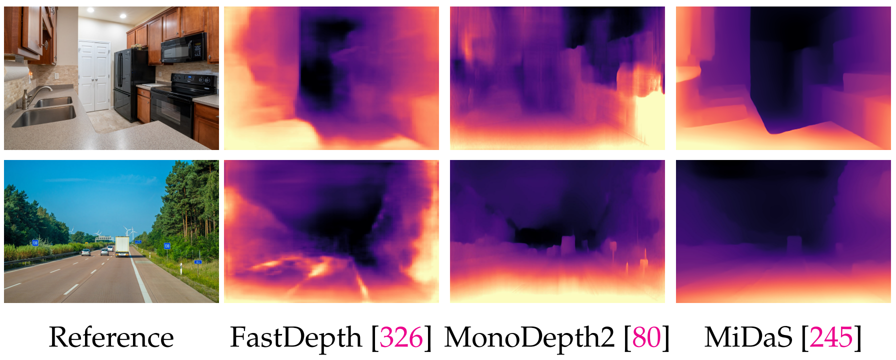


FIGURE 7.1: **Predictions in the wild.** We provide qualitative results for indoor and outdoor internet images, using checkpoints publicly available. It can be noticed how the networks trained on a single dataset, both in a supervised in indoor data (FastDepth) and self-supervised on outdoor driving sequences (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/`.

In this chapter, we describe how the teacher-student paradigm can help us in training any monocular network on a large amount of unlabeled data. Material and results presented in this chapter have been published in Real-time single image depth perception in the wild with handheld devices [12].

7.1 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. 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.

7.1.1 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, weakly-supervised or self-supervised training paradigms. Moreover, cheaper and better-scaling supervision can be conveniently obtained from another network, leveraging knowledge distillation to avoid the need for expensive ground-truth labels, through a teacher-student approach.

When a large enough dataset providing ground-truth labels inferred by an active sensor is available, such as [310], supervised training 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, [302, 321] exploit depth values from stereo algorithm, while Li et al. [161] rely on a SfM pipeline. Finally, supervision can be distilled from other networks as well, for the stereo [89] and monocular [229] setup. The latter is the strategy followed in this proposal. Specifically, we use as teacher the MiDaS network proposed in [245]. Notice that also Watson et al. [320] use MiDaS as supervisor, but in their case the student network is a stereo model and not a monocular one. 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 [245] would take much much longer time since mostly bounded by the demanding 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 [245], it is straightforward to distil 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.

7.1.2 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.

Identified and trained a suitable network, the mapping on a mobile device is nowadays quite easy. In fact, there exist various tools that, starting from a deep learning framework as PyTorch or TensorFlow, can export, optimize (e.g. perform weight quantization) and execute models even leveraging mobile GPUs [155] 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 iPhone devices can use the GPU or even the Apple Neural Engine (ANE) thanks to Metal and Metal Performance Shaders (MPS), thus largely improving the runtime performance.

7.2 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: PyD-Net [230], DSNet [9] and FastDepth [326]. Moreover, we also include a representative example of a larger network MonoDepth2, proposed in [80]. It is worth to notice that other and more complex state-of-the-art networks, as [302], 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 [302]).

MonoDepth2. This architecture is the MonoDepth2 (ResNet18) model [80], described in Chapter 3.3, in which we replaced nearest-neighbour feature interpolation with bilinear to avoid checkerboard artifacts. The network counts overall 14.84 million parameters. It is worth to notice that in our evaluation we do not rely on ImageNet [58] pre-training for the encoder for fairness to other architectures not pre-trained at all.

PyD-Net. We outlined the architecture of this model in Chapter 5. However, in this study we applied few changes with respect to the original network [230], e.g. transposed convolutions have been replaced by upsampling and convolution blocks to avoid checkerboard artifacts. Moreover, the network now has to predict a single disparity map aligned with the input image, since we do not need a *virtual* stereo setup. The overall network counts 1.97 million parameters.

FastDepth. We already introduce this network in Chapter 3.3. In this evaluation we do not rely on hardware-specific optimization strategies for fairness with other networks. The whole network counts 3.93 million parameters.

DSNet This architecture is part of ΩNet [9], the ensemble of networks introduced in Chapter 6. In our evaluation we consider only the depth estimation network

DSNet, which counts 1.91 millions of parameters, 0.2 millions fewer than the original model because the dedicated semantic head has been removed.

7.3 Wild dataset

The Wild dataset (W), consists of a mixture of Microsoft COCO [166] and OpenImages [147] 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 select almost 447,000 frames for training purposes. Then, we distilled the supervision required by our networks with the robust monocular architecture proposed in [245] with the weights publicly available. Such a network provides as output an inverse depth up to a scale factor. We point out once again that our supervision protocol has been carefully chosen mostly for practical reasons. It takes a few days to distil 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 [245], it would require an intensive effort and dedicated strategies for each type of data. For example, the authors followed the COLMAP [267] pre-processing of [163] for outdoor video sequences. For 3D movies, instead, the authors first removed videos not captured by real stereo cameras, then they divided each video into chapters and short clips; afterwards, each clip has been processed with optical flow and semantic segmentation neural networks to obtain the final labels. Notably, the optical flow network has been applied to both the images of each pair, in order to detect invalid pixels with the left-right consistency check.

On the contrary, our strategy 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 [245] 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 chapter.

7.4 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 [326] 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.

7.4.1 Evaluation on KITTI

At first, we investigate the accuracy of the considered networks on the KITTI dataset. Since the models have been developed with different frameworks (PyD-Net and DSNet in TensorFlow, the other two in PyTorch) and trained on different datasets (FastDepth on NYU v2 [206], others on the Eigen [65] split KITTI [196]), we implemented all the networks in PyTorch. This strategy allows us to adopt the same self-supervised protocol proposed in [80] 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 $I^\dagger$:

$$\hat{I} = \omega(I^\dagger, K^\dagger, R, T, K, \mathcal{D}) \tag{7.1}$$

where ω 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 [80] is composed of a photometric error term p_e and an edge-aware regularization term L_s. We already introduced p_e in 4.4, while L_s is the equation 4.5 in which the predicted depth for each pixel $\mathcal{D}_{ij}$ is replaced by the mean normalized inverse depth $\mathcal{D}^*_{ij} = \mathcal{D}_{ij}/\overline{\mathcal{D}}$. This strategy, initially proposed in [314], avoids the vanishing of the estimated depth by dividing the predicted map with its mean value $\overline{\mathcal{D}}$. We adopt the M configuration of [80] 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. Purposely, a camera pose network is trained to estimate relative camera poses between the frames in the sequence as in [80]. 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 [80]). 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 [80] 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 [139], 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. 640×192) 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 7.1 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 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

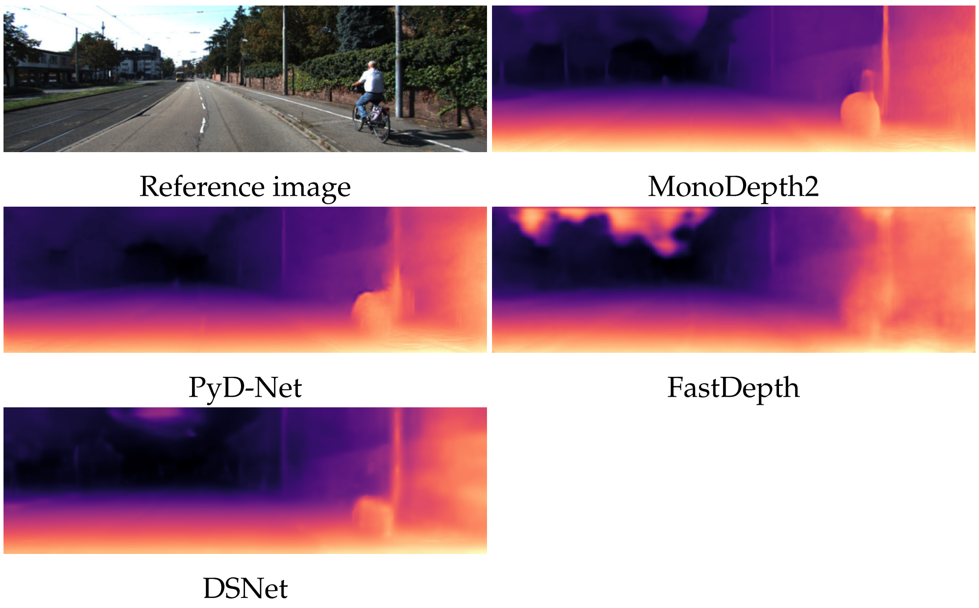


FIGURE 7.2: **Qualitative results on KITTI.** All the models have been trained equally using the framework of [80] on the Eigen split of KITTI.

their performance is comparable, PyD-Net results more effective with respect to FastDepth and DSNet on most metrics, such as RMSE and $\delta < 1.25$.

Figure 7.2 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.

Network	Abs Rel	Sq Rel	RMSE	log RMSE	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
PyD-Net	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 †	**0.132**	**1.044**	**5.142**	**0.210**	**0.845**	**0.948**	**0.977**
PyD-Net†	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 7.1: **Quantitative results on Eigen split.** † indicates models trained according to [80] training framework, otherwise we report results provided in each original paper.

7.4.2 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 7.1 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 (224×224 for [326], 1024×320 for [80]) 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 [80] 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. [245], 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 ($\sim$ 105M parameters) and very accurate monocular network.

We leverage this latter model to *distil* knowledge and train lightweight models compatible with mobile devices. As mentioned before, this strategy allows us to *use* MiDaS knowledge for faster training data generation compared to time-consuming pipelines used to train it, such as COLMAP [267, 268]. 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, clipping minimum and maximum between 1th and 99th percentile although this latter operation is not strictly required. Then, obtained such proxy labels, we train the networks using the following loss function:

$$\mathcal{L}(D_x^s, D_{gt}) = \alpha_l \left\| (D_x^s - D_{gt}) \right\| + \alpha_s \mathcal{L}_g(D_x^s, D_{gt}) \tag{7.2}$$

where $\mathcal{L}_g$ is a gradient loss term minimizing the absolute difference between the predicted depth derivatives (in both directions) and proxy depth derivatives, D_x^s the predictions of the network at scale s (bilinearly upsampled to full resolution) for the pixel x and D_{gt} is the proxy depth of x. The weight α_s depends on the scale s and is halved at each lower scale. On the contrary, α_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 640×320. We set the initial value of α_s to 0.5 for all networks except for FastDepth, set to 0.01. To augment the images, we applied random horizontal flip with 0.5 probability.

Table 7.2 collects quantitative results on three datasets, respectively TUM [284] (3D object reconstruction category), KITTI Eigen split [65] and NYU [206]. On each dataset, we first show the results achieved by large and complex networks MiDaS [245] and the model by Li et al. [161] (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. We adopted the same protocol defined in [245] to obtain depths at the same scale of ground-truth values from predictions. First and foremost, we highlight how MiDaS performs in general better than [161], emphasizing the reason to distil knowledge from it.

Considering lightweight compact models PyD-Net, DSNet and FastDepth we can notice that the margin between them and MiDaS is often non-negligible. A 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 7.2 is not so surprising. Looking more in details the outcome of lightweight networks, PyD-Net is the best model on

Network	Dataset	Abs Rel	Sq Rel	RMSE	log RMSE	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Ranftl et al. [245]	TUM	**0.125**	**0.148**	**0.832**	**0.195**	**0.857**	**0.944**	**0.978**
Li et al. [161]	TUM	0.135	0.158	0.852	0.209	0.826	0.942	0.975
MonoDepth2	TUM	0.147	0.180	0.916	0.222	0.811	0.927	0.967
PyD-Net	TUM	0.166	0.210	0.978	0.244	0.767	0.921	0.955
DSNet	TUM	0.168	0.215	0.994	0.248	0.762	0.917	0.951
FastDepth	TUM	0.160	0.209	0.982	0.241	0.780	0.918	0.955
Ranftl et al. [245]	KITTI	**0.157**	**1.144**	**5.672**	**0.225**	**0.780**	**0.942**	**0.980**
Li et al. [161]	KITTI	0.227	2.081	7.841	0.325	0.621	0.854	0.939
MonoDepth2	KITTI	0.164	1.194	6.000	0.239	0.752	0.928	0.974
PyD-Net	KITTI	0.162	1.272	6.138	0.239	0.760	0.927	0.974
DSNet	KITTI	0.164	1.203	5.977	0.239	0.754	0.929	0.975
FastDepth	KITTI	0.168	1.227	6.017	0.241	0.741	0.927	0.975
Ranftl et al. [245]	NYU	**0.100**	**0.061**	**0.407**	**0.132**	**0.905**	**0.984**	**0.997**
Li et al. [161]	NYU	0.149	0.116	0.560	0.189	0.782	0.958	0.992
MonoDepth2	NYU	0.123	0.082	0.473	0.160	0.848	0.974	0.995
PyD-Net	NYU	0.130	0.091	0.493	0.168	0.827	0.969	0.994
DSNet	NYU	0.134	0.096	0.505	0.171	0.820	0.968	0.993
FastDepth	NYU	0.129	0.090	0.492	0.167	0.833	0.971	0.994

TABLE 7.2: **Generalization on different datasets.** The three groups from top to bottom report experimental results concerned, respectively, with (top) TUM dataset, (middle) KITTI Eigen and (bottom) NYUv2.

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

TABLE 7.3: **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 640×384, averaged on 50 inferences.

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 7.3 shows a qualitative example when processing an online picture using MegaDepth [163], the model by Li et al. [161], MiDaS [245] and the networks trained through knowledge distillation.

Finally, in Figure 7.4, we also report two examples in which the student network inherits the failure of the teacher model MiDaS. Since both networks fail, the problem is not attributable to their different architecture. Observing these figures, we can notice that such behavior is due to an induced optical illusion (first case) or ambiguous objects, such as a mirror (second case).

7.4.3 Performance analysis on mobile devices

Once 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

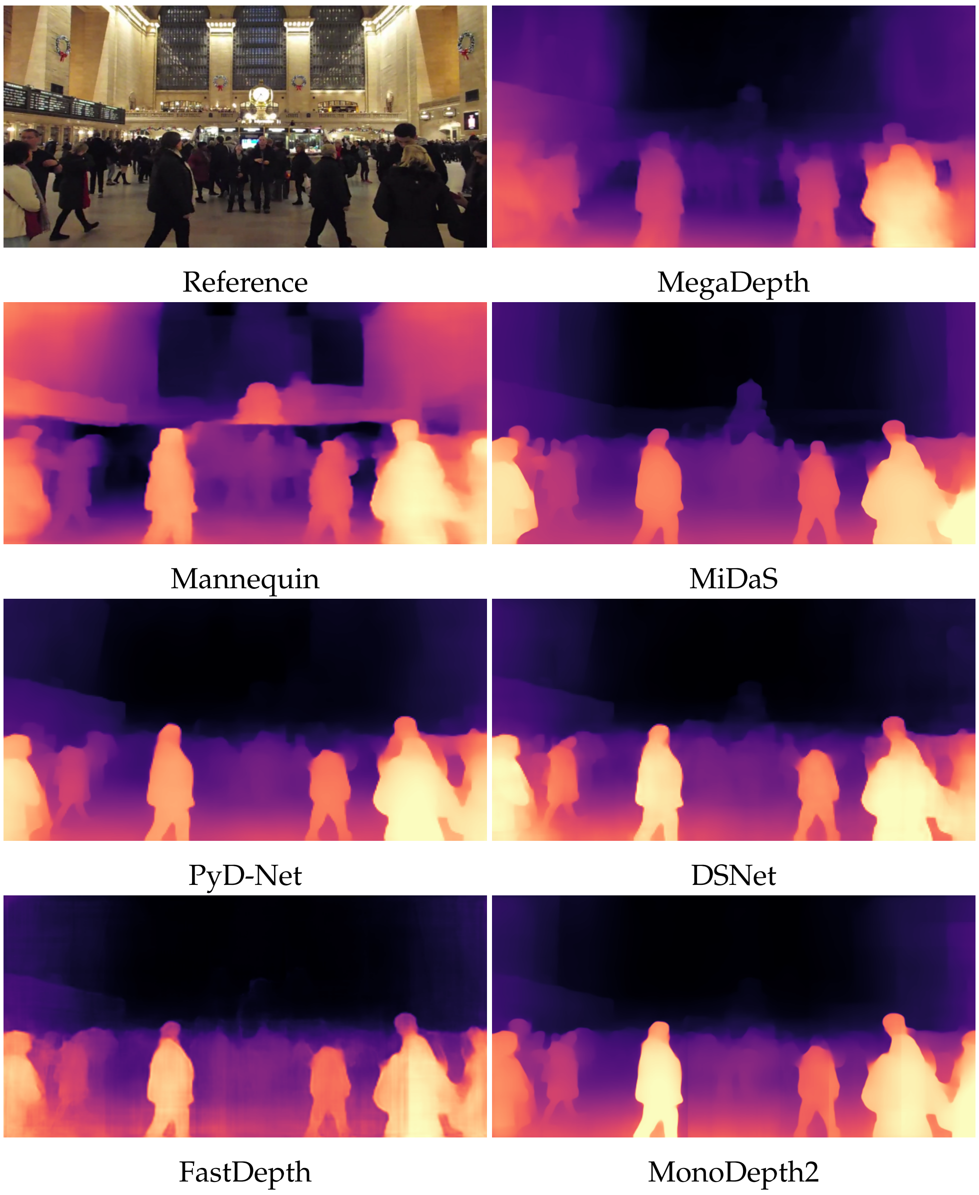


FIGURE 7.3: **Qualitative result from an online picture**. From left to right, the reference image from Pexels website, depths from MegaDepth [163], Mannequin [161], MiDaS [245], PyD-Net, DSNet, FastDepth and MonoDepth2.

only model conversion avoiding weights quantization not to alter the accuracy of the original network.

Table 7.3 collects stats about the considered networks. Specifically, we report the number of *multiply-accumulate* operations (MAC) and the frame rate (FPS) measured when deploying the converted models on an Apple iPhone XS. Measurements are gathered processing 640×384 images and averaging over 50 consecutive inferences. To the best of our ability, all the models can run on iPhone NPU except for MiDaS, which is not able in our tests to leverage such accelerator. 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

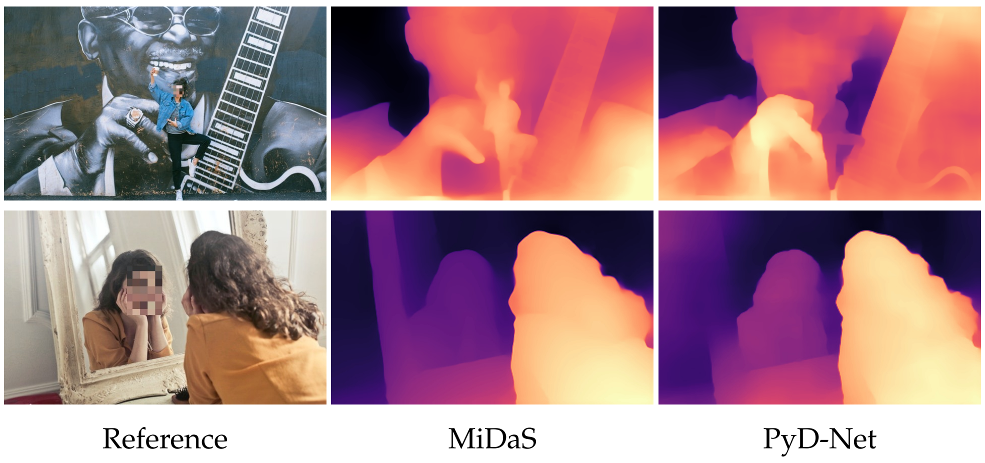


FIGURE 7.4: **Failure cases.** Examples of single image depth estimation failures. From left to right: input image, depth predicted by the teacher MiDaS and by the student PyD-Net.

performing one order of magnitude fewer operations. DSNet and PyD-Net 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 PyD-Net, 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 PyD-Net when deployed with the same degree of optimization of the other networks on the iPhone XS.

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 PyD-Net as the best trade-off between accuracy and speed when targeting handheld devices.

7.5 Applications of single image depth estimation

Assessed the performance of the networks, we present two well-known applications that can significantly take advantage of dense single image depth estimation. For these experiments, we use the PyD-Net 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 τ, all the pixels with a relative inverse depth smaller than τ are blurred by a 25×25 Gaussian kernel. In Figure 7.5, we apply the bokeh effect to single images sampled from the web, for which neither stereo pairs nor video sequences are available.

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 using camera tracking with respect to fixed a reference system located in an *anchor point*. This goal is achieved by matching sparse feature points and taking advantage of the sensor suite (comprising accelerometers, gyroscope, etc) available mobile devices. An additional outcome of this process is a bunch of sparse depth measurements appropriately scaled. The leftmost image

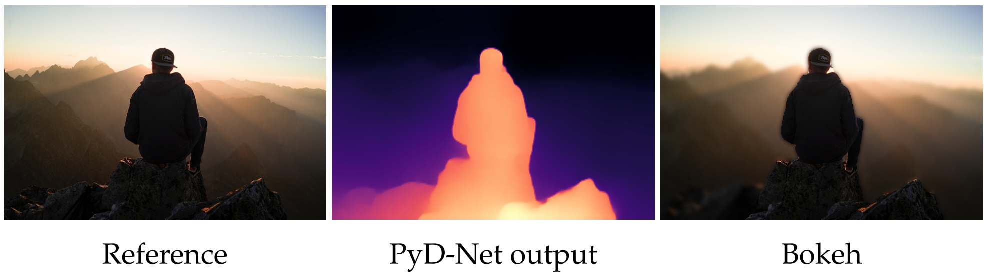


FIGURE 7.5: **Bokeh effect.** Given the reference image, we smooth farther pixels in the image using depth values provided by PyD-Net.

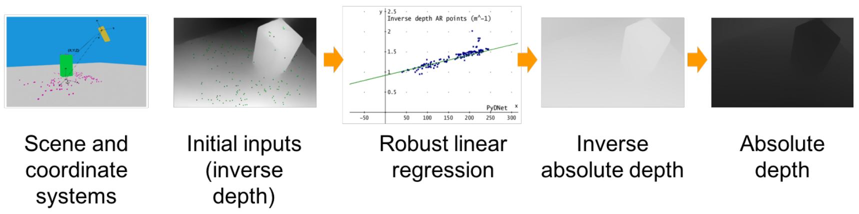


FIGURE 7.6: **AR Pipeline.** The leftmost figure depicts an object (green box) framed by a device (yellow box) capable of providing sparse absolute depth points (in purple) through the AR framework and a dense inverse depth map thanks to the monocular network. To realize our improved AR application, we move first sparse depth measurements into the inverse depth domain (second image from the left). Then, we obtain the shift and scale factor that maps such sparse points and corresponding predictions picked by the dense inverse map provided by the network by robustly regressing a linear model within a RANSAC framework, as illustrated in the chart. Finally, we scale the dense inverse map accordingly before turning back into the depth domain (respectively, the two rightmost maps) to render virtual objects consistently to the inferred geometry of the scene.

of Figure 7.6 illustrates the standard output and reference system of a standard AR framework.

However, these frameworks 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 physic interactions with the real world, such as occlusions. Although some authors proposed to densify the sparse depth measurements provided by AR frameworks, it is worth observing that dynamic objects or other factors in the sensed scene may lead to incorrect depth estimations [103].

On the other hand, 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 leveraging the sparse absolute depth measurements provided by standard AR framework. Purposely, we developed a mobile application capable of handling object occlusions by combining sparse clues provided by standard AR frameworks, such as ARCore or ARKit, to scale accordingly at each frame the dense depth prediction provided by a lightweight monocular network. To achieve this goal, at first, we convert the sparse absolute depth measurements inferred by the AR framework into inverse depths to be compliant with the output domain of the monocular network, encoding an inverse depth up to a scale and a shift factors. Then, within a RANSAC framework, we regress the parameters of a linear model enabling to scale the whole network's

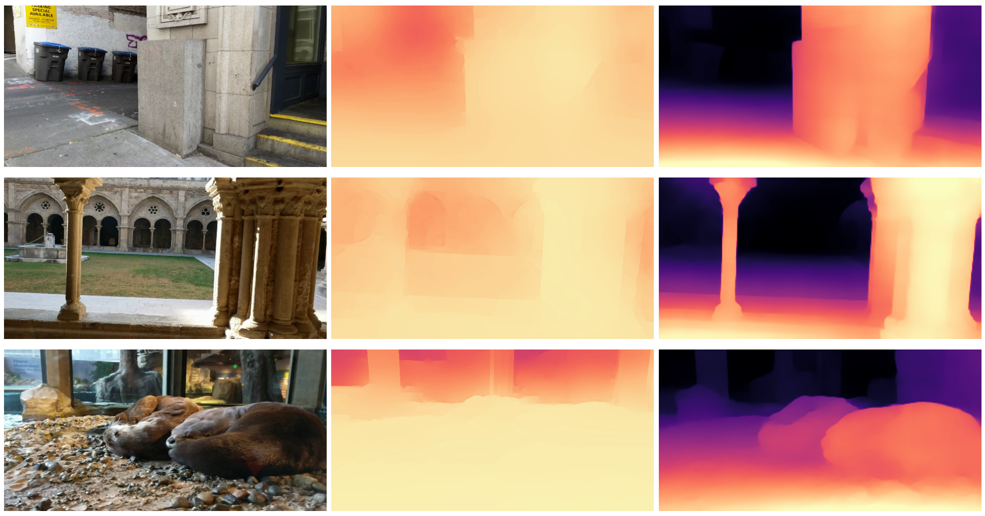


FIGURE 7.7: **Qualitative comparison with other occlusion-aware AR methods.** From left to right, the input image, the depth from [103] and PyD-Net predictions.

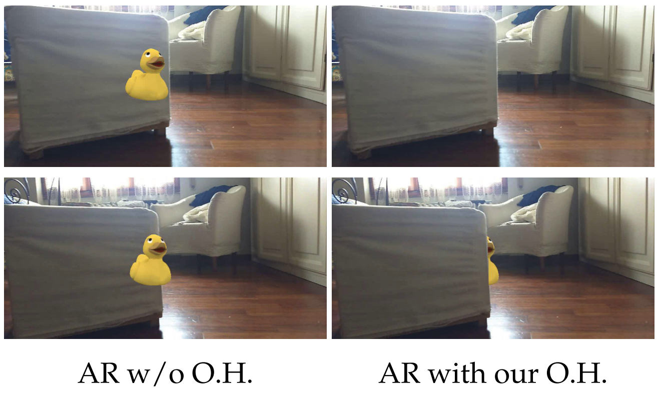


FIGURE 7.8: **AR with occlusion handling (O.H.).** On the left, vanilla AR enabled by an Android device with ARCore. On the right, instead, our depth-aware AR enabled by single image depth prediction with PyD-Net for occlusion handling.

output according to the sparse, yet scaled inverse depth predictions. RANSAC is adopted to take into account possible outliers in predicted maps that can ruin dramatically the goodness of the model. Specifically, for multiple times (e.g. 100) we sample a subset of the points to estimate the model, and we use the remaining ones for verification; after the last iteration, we select the model that achieved the best verification score. As the last step we turn inverse depths into depths, obtaining absolute depth predictions enabling rendering of virtual objects consistent with the geometry of the scene. The overall pipeline outlined is illustrated in figure 7.6.

Differently from other approaches, such as [103] and [180], 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 two points in scale suffice to obtain absolute dense 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 7.7 how our strategy coupled with PyD-Net can produce competitive and detailed depth maps leveraging a single RGB image only. Figure 7.8 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 PyD-Net and sparse anchors provided by a conventional AR framework.

7.6 Conclusion

In this chapter we have introduced a fast and easy-to-apply strategy to achieve robust monocular depth estimation on lightweight handheld devices characterized by severe constraints concerning power consumption and computational resources. To achieve this goal, we first distill from a set of single still images the knowledge from a pre-trained state-of-the-art robust network unsuited for real-time execution yet capable to generalize very well on new environments. Then, we train and evaluate lightweight monocular models, unlocking real-time performance even on mobile devices. The evaluation highlights that effective real-time depth estimation in the wild from a single image is possible on consumer devices by adopting the framework outlined in this chapter. As further proof of this achievement, we have shown its deployment in two relevant consumer applications.

Chapter 8

Distilling optical flow labels using monocular depth

In Chapters 4,5 and 6 we have introduced novel frameworks aimed at infer monocular depth, while in chapter 7 we have seen how we can distill with a very limited effort thousands of labels using off-the-shelf monocular models. The availability of robust monocular depth models is nowadays the state of affairs, and they are paving the way for new applications and purposes. Some examples are the bokeh and the improved augmented reality presented in Chapter 7. Furthermore, Watson et al. [320] show that these models are also pivotal components for training stereo models on large unlabeled datasets with real texture. In their method, the monocular network is not the final objective but it represents a support for a larger plan. Following this rationale, can we adopt off-the-shelf robust monocular models for generating labels in other fields, such as optical flow?

As introduced in Chapter 1, optical flow task consists in estimating the 2D motion of pixels between two frames. Traditional methods, as [105, 29], paved the way to preliminary solutions for the problem, allowing to realize applications as frame interpolation or motion tracking. Deep learning boosted the accuracy on conventional benchmarks [77, 196, 37, 34], with supervised training strategies still representing the best way to obtain accurate models. Unfortunately, sourcing labels for this task is even more challenging than for depth estimation, since no real sensor is able to measure directly such values. Unsupervised training strategies relaxed this constraint, allowing to train the model on unlabeled images, but they do not provide a strong supervision on difficult regions, such as texture-less areas or repetitive patterns.

In this chapter we introduce a framework, called *depthstillation*, to generate accurate ground-truth optical flow annotations quickly and in large amounts from any readily available single real picture. Specifically, given an image we use an off-the-shelf monocular depth estimation network to build a plausible point cloud for the observed scene. Then, we virtually move the camera in the reconstructed environment with known motion vectors and rotation angles, allowing us to synthesize both a novel view and the corresponding optical flow field connecting each pixel in the input image to the one in the new frame. When trained with our data, state-of-the-art optical flow networks achieve superior generalization to unseen real data compared to the same models trained either on annotated synthetic datasets or unlabeled videos, and better specialization if combined with synthetic images.

The content of this chapter has been published in Learning optical flow from still images [7].

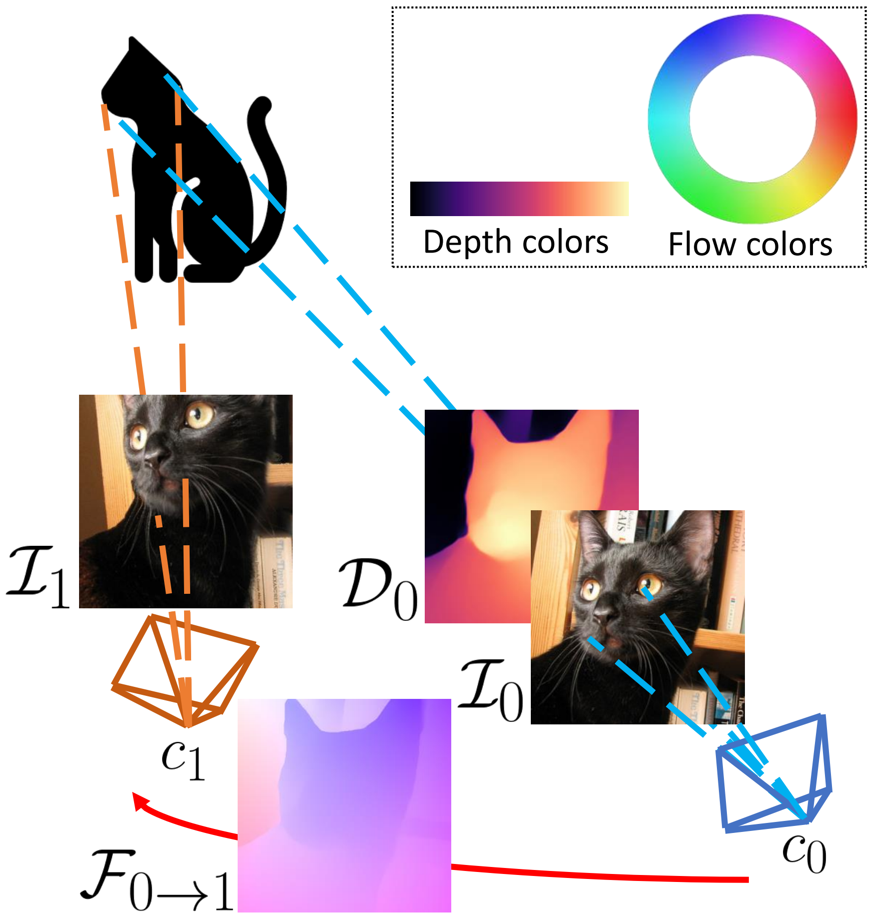


FIGURE 8.1: **Overview of the proposed depthstillation pipeline.** Given a single image $\mathcal{I}_0$ and its estimated depth map $\mathcal{D}_0$, we place the camera in c_0 and *virtually* move it (red arrow) towards a new viewpoint c_1. From the depth and virtual ego-motion, we obtain optical flow labels $\mathcal{F}_{0\rightarrow 1}$ and a novel $\mathcal{I}_1$ through forward warping.

8.1 Depthstillation pipeline

In this section we illustrate our proposed framework to *depthstill* a training sample, i.e. how we generate new virtual views $\mathcal{I}_1$ from single images $\mathcal{I}_0$, with corresponding dense optical flow ground-truth maps $\mathcal{F}_{0\rightarrow 1}$. An overview of our pipeline is shown in Figure 8.1.

Given $\mathcal{I}_0$, an off-the-shelf monocular depth network Φ estimates the corresponding depth map $\mathcal{D}_0$

$$\mathcal{D}_0 = \Phi(\mathcal{I}_0) \tag{8.1}$$

Then, $\mathcal{D}_0$ is used to back-project pixels of $\mathcal{I}_0$ to their 3D points according to some plausible inverse intrinsics matrix K^{-1}. In case the network estimates inverse depth, we bring it to the depth domain first. $\mathcal{D}_0$ usually shows blurred edges [320, 275], causing flying pixels in the 3D space that can be easily sharpened via edge-preserving filters [184].

We now assume the camera used to frame image $\mathcal{I}_0$ to be at 3D location c_0 and apply an arbitrary *virtual* motion, moving it towards a new position c_1. To this aim, we generate a plausible rotation R_1 by sampling a random triplet of Euler angles and a plausible translation t_1 by sampling a random 3D vector. Then, we obtain the

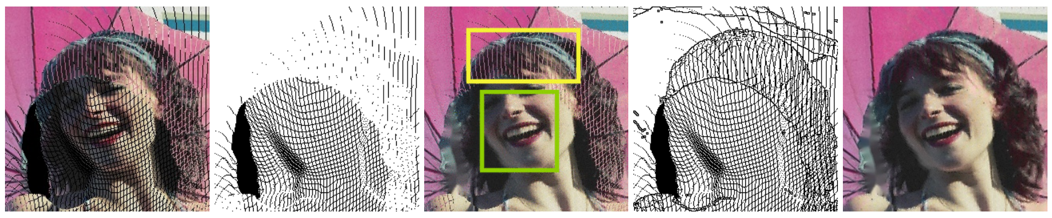


FIGURE 8.2: **Hole filling strategies.** From left to right: a) forward-warped image affected by stretching artefacts, b) holes mask $\mathcal{H}$ c) inpainted image, d) collision-augmented holes mask $\mathcal{H}'$ and e) improved inpainted image. Black pixels in $\mathcal{H}$ and $\mathcal{H}'$ are those to be inpainted. Reference picture from [166].

transformation matrix $T_{0\rightarrow 1} = (R_1|t_1)$ corresponding to such roto-translation. Thus, we can project our 3D points to the image space through K in order to obtain a new image $\mathcal{I}_1$. This allows to obtain, for each pixel p_0 in $\mathcal{I}_0$, the coordinates p_1 of its corresponding pixel in $\mathcal{I}_1$ acquired from viewpoint c_1

$$p_1 \sim KT_{0\rightarrow 1}\mathcal{D}_0(p_0)K^{-1}p_0 \tag{8.2}$$

and flow $\mathcal{F}_{0\rightarrow 1}$ is obtained as the difference between p_1 and p_0.

As described in Chapter 6.3, $\mathcal{F}_{0\rightarrow 1}$ only models the virtual camera ego-motion, i.e. no object has moved independently. Finally, we obtain the new image $\mathcal{I}_1$ through forward warping.

Forward warping suffers from two well-known problems [320], that are *collisions* (i.e. multiple pixels from $\mathcal{I}_0$ being warped to the same location in $\mathcal{I}_1$) and *holes* (i.e. pixels in $\mathcal{I}_1$ over which no pixel from $\mathcal{I}_0$ is projected). To handle collisions, we keep track of pixels p_1 having multiple projections p_0 in a binary collision mask $\mathcal{M}$ (i.e. collisions are labeled as 1, other pixels as 0) and select, for each, the one having minimum depth according to camera in position c_1, i.e. the closest, to be displayed in $\mathcal{I}_1$.

Hole filling. Artefacts introduced by holes are more subtle to be solved. Moreover, applying a 6DoF transformation to the camera plane vastly increases the chance of occurrence of holes compared to the case of 1D camera translations applied to distill stereo pairs [320]. In particular, in case of larger camera motion/rotations some *stretching* artefacts occur on the foreground objects (and, occasionally, in the background as well) as shown in Figure 8.2 a). To remove these holes, we build a binary hole mask $\mathcal{H}$, as in Figure 8.2 b), where we label pixels in $\mathcal{I}_1$ for which no pixel in $\mathcal{I}_0$ is reprojecting on to with 0. Then, a simple inpainting strategy [295] is usually sufficient to fill them, as reported in Figure 8.2 c) on the girl's face (green rectangle).

Unfortunately, this is not enough in the case of stretching artefacts occurring in a foreground object overlapping a background one. Indeed, in this case, it is very likely that the holes induced by the stretching of the foreground object are filled by pixels in the background. These pixels are not detected by $\mathcal{H}$, causing the bleeding effect shown in the yellow rectangle of Figure 8.2 c), where the hair merges with the background umbrella. Since most of these artefacts occur in non-colliding pixels surrounded by colliding ones, i.e. in $\mathcal{M}$ they are labeled as 0 and surrounded by pixels labeled as 1, we can detect them by dilating $\mathcal{M}$ into $\mathcal{M}'$. Then, we define the binary mask $\mathcal{P}$ assigning 1 to pixels having the same label in $(\mathcal{M}', \mathcal{M})$ and 0 to the remaining (i.e. those that become 1 in $\mathcal{M}'$). We finally obtain $\mathcal{H}'$ by multiplying $\mathcal{H}$

and $\mathcal{P}$. In other words, for every pixel x we obtain the following equations:

$$\mathcal{P}(x) = \begin{cases} 1 & \text{if} \quad \mathcal{M}'(x) = \mathcal{M}(x) \\ 0 & \text{otherwise} \end{cases} \qquad \mathcal{H}'(x) = \mathcal{H}(x) \cdot \mathcal{P}(x) \tag{8.3}$$

We can apply the inpainting algorithm to pixels labeled with 0 in $\mathcal{H}'$, shown in Figure 8.2 d), to obtain Figure 8.2 e), where the foreground-background bleeding does not occur.

We point out how, in large dis-occluded area (i.e., in the proximity of depth boundaries, as shown in Figure 8.2 on the left of the person), the inpainting method produces blurred content, as shown in Figure 8.2 c) and e). Despite these artefacts, our experiments will prove that hole filling improves the accuracy of trained networks significantly.

Independent motions. The pipeline sketched so far models the optical flow field occurring between images acquired in a static environment, i.e. consequence of the camera motion, not taking into account possible independently moving objects, very likely to occur in real contexts [196]. In order to model more realistic simulations, we introduce the possibility of applying different virtual motions to objects extracted from the scene by leveraging an instance segmentation network Ω for extracting N objects Π_i, $i \in$ [1,N]

$$\Pi = \{\Pi_i, i \in [1, \text{N}]\} = \Omega(\mathcal{I}_0) \tag{8.4}$$

Then, to simulate a motion of the object in the scene, we randomly move the camera from c_0 towards a point $c_{\pi_i} \neq c_1$ and its corresponding transformation $T_{0\to\pi_i}$ to be applied to object Π_i. Then, we reproject pixels from $\mathcal{I}_0$ on the image planes of the different cameras. Pixel coordinates in $\mathcal{I}_1$ will be selected according to their belonging to segmented objects or the background as:

$$p_1 \sim \begin{cases} KT_{0\to1}\mathcal{D}_0(p_0)K^{-1}p_0 & \text{if } p_0 \notin \Pi \\ KT_{0\to\pi_i}\mathcal{D}_0(p_0)K^{-1}p_0 & \text{if } p_0 \in \Pi_i \end{cases} \tag{8.5}$$

We handle collisions as outlined before, keeping pixels whose depth results lower after motion. Finally, we obtain optical flow $\mathcal{F}_{0\to1}$ and image $\mathcal{I}_1$ as aforementioned.

To be robust to noisy/false detections, e.g. in case of tiny blobs accidentally labeled as objects, we rank the objects according to their size, i.e. number of pixels, and keep in Π only the $n <$N largest objects. Figure 8.3 shows two qualitative comparisons between images and flow distilled by merely applying a virtual camera motion, a) and b), and those obtained by segmenting the cat or the person in the foreground and simulating an independent motion, c) and d). Although our formulation simulates moving objects by moving virtual cameras instead, we can notice how the final effect on $\mathcal{I}_1$ and $\mathcal{F}_{0\to1}$ is equivalent for our purposes.

We point out that, by increasing the number of moving objects, collisions and holes increase. In particular, a higher number of dis-occlusions might appear after applying independent motions, leading to blurry inpainted content, as shown in Figure 8.3 c) on the top row, on the right of the cat. Besides, shape boundaries may be inconsistent across depth and segmentation predictions, afflicting the truthfulness of the generated image and introducing artefacts (e.g., background pixels moved as part of the foreground). We will see how, although helpful, this approach yields minor improvements compared to the previous two steps performed in our

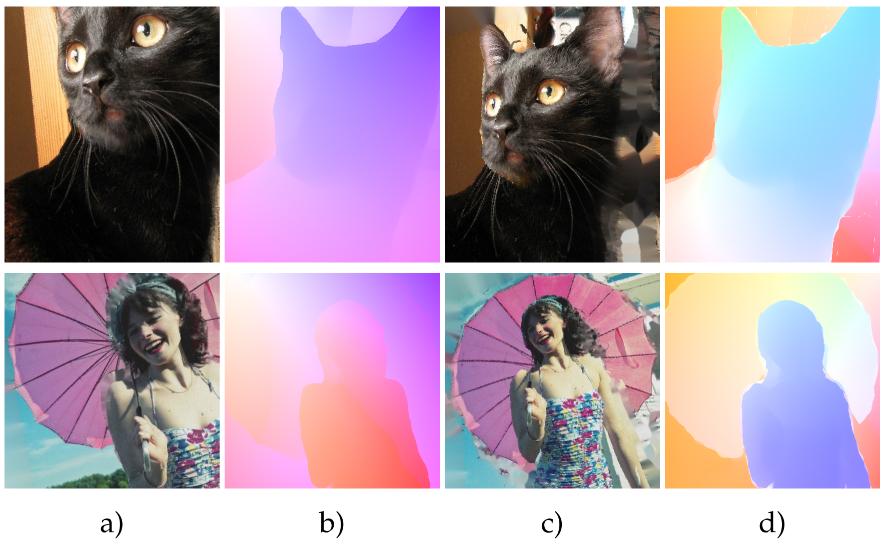


FIGURE 8.3: **Independent motions modeling.** From left to right: a) image generated by only modeling camera motion and b) corresponding optical flow field, c) image generated after segmenting the foreground, which is now subject to a different motion yielding d) a more complex optical flow field.

framework, that result crucial for dephtstilling reliable training data. Moreover, segmenting object instances requires an additional network Ω trained in a supervised manner conversely to single-image depth estimation networks, whereby an extensive literature of self/weakly-supervised approaches exists [79, 80, 302, 321].

8.2 Experimental results

In this section, we describe the experimental setup used to validate our depthstillation pipeline.

8.2.1 Training datasets

At first, we describe the datasets used to train the networks considered in our experiments.

FlyingChairs (Ch). We refer to Chapter 3.1 for the description of this dataset.

FlyingThings3D (Th). Already presented in Chapter 3.1, the FlyingThings3D dataset [120] is a collection of 3D synthetic scenes belonging to the SceneFlow dataset [191] and contains a training split made of 19, 635 images. Differently from Chairs, objects move in the scene with more complex 3D motions. State-of-the-art networks usually train in sequence over Chairs and Things (Ch$\rightarrow$Th).

COCO dataset. The COCO dataset [166] is a collection of single still images (it provides $\mathcal{I}_0$ only) and ground-truth with labels for tasks such as object detection or panoptic segmentation, but lacks any depth or optical flow annotation. We sample images from the *train2017* split, which contains 118, 288 pictures, to generate virtual images and optical flow maps. We dub *dephtstilled* COCO (**dCOCO**) the training set obtained in such a manner.

DAVIS. The DAVIS dataset [223] provides high-resolution videos and it is widely used for video object segmentation. Since it does not provide optical flow ground-truth labels, we use all the 10, 581 images of the unsupervised 2019 challenge to generate **dDAVIS** and compare with the state-of-the-art in self-supervised optical flow [130].

8.2.2 Testing datasets

We describe here the testing imagery used to evaluate the networks trained on the datasets mentioned above. We report the EPE, > 3 and the Fl metrics, defined in Chapter 3.2, on *All* pixels. In every experiment, we will highlight the best results in **bold** and underline the second-best among methods trained in fair conditions. We evaluate models on Sintel (Clean and Final passes) [37], KITTI 2012 [77] and KITTI 2015 [196]. These datasets have already been introduced in Chapter 3.1.

8.2.3 Implementation details

We describe next our pipeline and the networks used for depth estimation and learning optical flow.

Depth estimation models. To obtain dense depth maps from single RGB images, we select two models, respectively MiDaS [245] and MegaDepth [163], the former because represents the state-of-the-art for depth estimation in-the-wild and the latter because trained with weaker supervision than MiDaS[1]. Next, we will show how the accuracy of networks trained on our data is affected by the depth estimator.

Depthstillation pipeline. To generate virtual images, we convert predicted depths into $[1, 100]$. Given a single image of resolution W×H, we assume a virtual camera having fixed K, with focals $(f_x, f_y) = 0.58$(W,H) and optical center $(c_x, c_y) =$ 0.5(W,H). To generate $T_{0\rightarrow 1}$, we build t_1 by sampling three scalars t_x, t_y, t_z in $[-0.2, 0.2]$ and R_1 by sampling three Euler angles in $[-\frac{\pi}{18}, \frac{\pi}{18}]$. To simulate moving objects, we run pre-trained Mask-RCNN [97] to select $n = 2$ instance masks and generate t_i and R_i sampling respectively in $[-0.1, 0.1]$ and $[-\frac{\pi}{36}, \frac{\pi}{36}]$ and add them to R_1 and t_1. Depth maps are sharpened by means of 2 iterations of a 5×5 bilateral filter, while we dilate $\mathcal{M}$ with a 3×3 kernel. We can generate multiple camera motions for any given single image and thus a variety of pairs and ground-truth labels. We will see how playing with the number of images and motions impacts optical flow network accuracy.

Optical Flow networks. To evaluate how effective our distilled images are at training optical flow models, we select two main architectures: RAFT [294] and PWC-Net [287]. The first because it represents state-of-the-art architecture for supervised optical flow, already enabling excellent generalization capability. The second because it achieves the best results among self-supervised methods (e.g. UFlow [130]). By deploying both architectures, we aim to prove that our method is general and significantly improves generalization in supervised and self-supervised optical flow. When not otherwise specified, we train RAFT on depthstilled data for 100K steps with a learning rate of 4×10^{-4} and weight decay of 10^{-4}, batch size of 6 and 496×368 image crops. This configuration is the largest one fitting into a single NVIDIA Titan X GPU. Following [294], we adopted AdamW as optimizer [175] and applied the same data augmentations and loss functions, while we set 12 as the number of iterative updates. To train PWCNet, we used as optimizer Adam [139]

[1]The reader might argue that MiDaS has been trained on labels produced by pre-trained optical flow networks, introducing biases into images generated with our pipeline. However, we point out that optical flow networks are used only to handle negative disparities in stereo images and would not be necessary if, given the minimum negative disparity d_{min}, the right image is shifted left by $|d_{min}|$, thus making $d_{min} = 0$.

as in the original paper [287], with an initial learning rate of $1e^{-4}$ and halved after 400K, 600K and 800K steps. We trained our model for 1M steps with a batch size of 8, adopting the multi-scale loss used in [287] for the synthetic pre-training, with the same augmentations and crop size used for RAFT.

8.2.4 Ablation study

In this section, we assess the impact of the different components of our pipeline.

	Depth	Hole	Moving	Sintel C.		Sintel F.		KITTI 12		KITTI 15	
	est.	fill.	obj.	EPE	> 3	EPE	> 3	EPE	Fl	EPE	Fl
(A)	✗	✗	✗	5.50	18.22	6.08	20.83	3.31	18.95	10.51	35.52
(B)	✓	✗	✗	<u>2.52</u>	7.17	<u>3.72</u>	<u>11.04</u>	2.02	7.53	4.84	16.26
(C)	✓	✓	✗	2.63	<u>7.00</u>	3.90	11.31	**1.82**	<u>6.62</u>	<u>3.81</u>	<u>12.42</u>
(D)	✓	✓	✓	**2.35**	**6.11**	**3.62**	**10.10**	<u>1.83</u>	**6.53**	**3.65**	**11.98**

TABLE 8.1: **Method ablation.** We train RAFT on dCOCO with different configurations of depthstillation: (A) constant depth for each image, (B) adding depth estimated by MiDaS [245], (C) adding hole-filling and (D) simulating object motions.

Depth, hole filling and moving objects. We start by ablating our pipeline to measure the impact of i) estimating depth, ii) applying hole filling to generated images and iii) simulating objects moving independently. This study is carried out by generating virtual views from 20K COCO images, applying a single virtual camera motion for each, by training RAFT [294] on them and evaluating the final model on Sintel, KITTI 2012 and KITTI 2015. Table 8.1 collects the outcome of this evaluation. On row (A), we show the performance achieved by generating images without estimating their depth, thus assuming a constant depth value for all pixels in any image. By moving to row (B), for which we use MiDaS [245] to estimate depth during the depthstillation process, we can notice considerable improvements in all metrics and datasets, with Fl score often more than halved. Nonetheless, generated images are affected by large holes and this does not allow for optimal performance. By enabling hole filling (C), the trained RAFT further improves its accuracy on real datasets. Finally, in (D), we show results by simulating objects moving independently, that further improves the results on Sintel. The benefit of this latter strategy is consistent on most metrics, although minor on real datasets such as KITTI 2012 and 2015 compared to the improvements obtained by (B) and (C), proving that the simple camera motion combined with depth is enough to obtain a robust optical flow network capable of generalizing to real environments. Moreover, as already pointed out, (D) also requires a trained instance segmentation network, which is hard to obtain for any possible dataset and would consequently constrain our pipeline. Thus, since our primary focus is on real environments, we choose (C) as the configuration for the following experiments.

Depth estimation network. We measure the impact of the depth estimator on our overall data generation pipeline. To this aim, we follow the same protocol of the previous experiments, replacing MiDaS with MegaDepth [163] during the depth estimation step. Table 8.3 shows the results of this experiment. We can notice how images generated through MegaDepth (B) allow for training a RAFT model that places in between the one trained on images generated without depth (A) and using MiDaS (C), being much closer to the latter than to the former. This proves that depth

	# Training samples			Sintel C.		Sintel F.		KITTI12		KITTI15	
	Images	Motions	Total	EPE	>3	EPE	>3	EPE	Fl	EPE	Fl
(A)	4K	$\times 1$	4K	2.73	6.96	3.97	11.09	1.86	6.81	3.93	12.56
(B)	4K	$\times 5$	20K	2.56	6.78	3.88	10.99	**1.77**	**6.62**	3.93	12.57
(C)	20K	$\times 1$	20K	2.63	7.00	3.90	11.31	1.82	**6.62**	**3.81**	12.42
(D)	20K	$\times 5$	100K	**2.37**	**6.69**	**3.64**	**10.73**	1.79	6.79	3.82	**12.39**

TABLE 8.2: **Impact of images and virtual motions.** We train several RAFT models by changing the number of input images taken from COCO and the number of motions depthstilled for each one.

is a crucial cue in our pipeline and the accuracy of the optical flow network, as we might expect, increases with the quality of the estimated depth maps, although with minor gains.

	Depth Model	Sintel C.		Sintel F.		KITTI12		KITTI15	
		EPE	>3	EPE	>3	EPE	Fl	EPE	Fl
(A)	No depth	5.50	18.22	6.08	20.83	3.31	18.95	10.51	35.52
(B)	Megadepth [163]	2.91	7.51	3.99	11.55	**1.81**	7.11	4.10	13.70
(C)	MiDaS [245]	**2.63**	**7.00**	**3.90**	**11.31**	1.82	**6.62**	**3.81**	**12.42**

TABLE 8.3: **Impact of depth estimator.** We train RAFT on dCOCO without depth estimation (A), using depth maps provided by MegaDepth (B) or MiDaS (C).

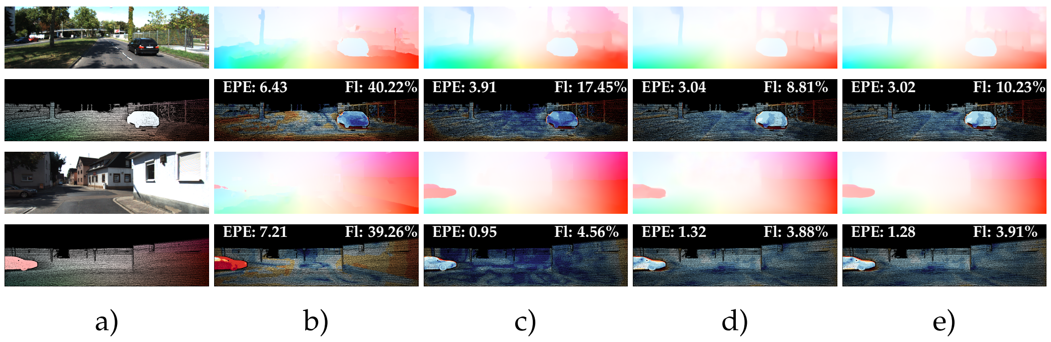


FIGURE 8.4: **Qualitative results on the KITTI 2015 training set.** On two rows: a) reference frame (top) and ground-truth flow (bottom), optical flow maps (top) by RAFT trained on b) Ch, c) Ch→Th, d) dCOCO and e) Ch→Th→dCOCO and error maps (bottom).

Amount of generated images. We can increase the amount of data we generate acting on two orthogonal dimensions: the number of images $\mathcal{I}_0$ and the number of virtual motions we simulate for each. Table 8.2 collects the results achieved by several RAFT models trained on a different number of images, obtained by varying the parameters mentioned above. By assuming 4K input images, we can notice how applying 5 virtual motions to each (B) allows a consistent boost on Sintel and KITTI 2012 compared to simulating a single motion each (A), while not improving on KITTI 2015. Interestingly, 4K images already allow for strong generalization to real domains, outperforming the results achieved using synthetic datasets shown in detail in the next section. On the other hand, increasing the input images by the same factor $\times 5$, yet simulating a single motion (C) leads worse results on Sintel while achieving some improvement on KITTI compared to (A) and (B). This fact highlights that a more variegate image content in the training dataset may be beneficial only for generalization to real environments. By depthstilling 5 motions, for a total of 100K training samples (D), yields further improvements on Sintel, again with minor impact on KITTI. To carry out a fair comparison with synthetic datasets, counting about 20K images each, we will use 20K images and a single virtual motion to depthstill our training data from now on.

8.2.5 Comparison with synthetic datasets

In this section, we evaluate the effectiveness of our depthstilled data versus synthetic datasets [61, 191].

Generalization to real environments. We start by evaluating the robustness of a network trained on our data when deployed on real datasets. Table 8.4 shows the performance achieved by RAFT when trained on Chairs (A) and fine-tuned on Things (B) with crop size and settings described in [294] to fit in a single GPU, compared to a variant trained on dCOCO, a split of 20K image pairs depthstilled from COCO (C). For completeness, we also report the performance of RAFT models provided by the authors (A†) and (B†), trained on $2\times$ GPUs and thus not directly comparable with our setting. We can notice how training on dCOCO (C) allows for much higher generalization on real datasets such as KITTI 2012 and 2015, at the cost of worse performance on the Sintel synthetic dataset. This latter result is not surprising because the images in Things are generated through computer graphics as those in Sintel, while generating virtual images from a real dataset (COCO) leads to superior generalization on real datasets (KITTI 2012 and 2015), also outperforming (A†) and (B†) despite the single GPU.

	Dataset	Sintel C.		Sintel F.		KITTI 12		KITTI 15	
		EPE	>3	EPE	>3	EPE	Fl	EPE	Fl
(A†)	Ch	2.26	7.35	4.51	12.36	4.66	30.54	9.84	37.56
(B†)	Ch→Th	1.46	4.40	2.79	8.10	2.15	9.30	5.00	17.44
(A)	Ch	2.36	7.70	4.39	12.04	5.14	34.64	10.77	41.08
(B)	Ch→Th	**1.64**	**4.71**	**2.83**	**8.67**	2.40	10.49	5.62	18.71
(C)	dCOCO	2.63	7.00	3.90	11.31	1.82	**6.62**	3.81	**12.42**
(D)	Ch→Th→dCOCO	1.88	5.31	3.23	9.26	**1.78**	7.00	**3.42**	13.08

TABLE 8.4: **Comparison with synthetic datasets – generalization.** Generalization achieved by RAFT when trained on synthetic data (A) and (B), on our dCOCO dataset (C) and a combination of both (D). † are obtained with publicly available weights by [294] ($2\times$ GPUs).

We also train RAFT sequentially on Chairs, Things and dCOCO (D). This setting improves the EPE achieved by (C) on KITTI 2012 and 2015 and turns out much more effective on Sintel with both metrics. This fact suggests that a combination of synthetic images with perfect ground-truth and virtual images with depthstilled labels might be beneficial for generalization purposes. Figure 8.4 shows some qualitative optical flow predictions and corresponding error maps obtained from the RAFT variants considered in Table 8.4.

Fine-tuning on real data. We evaluate the effect of pre-training on synthetic images or our generated frames when fine-tuning on a few real data with accurate ground-truth. To this aim, we fine-tune RAFT variants on the first 160 images of the KITTI 2015 training set and evaluate on the remaining 40 and KITTI 2012. We train with a learning rate of 10^{-4} and weight decay of 10^{-5}, batch size of 3 and 960×288 image crops, converging after 20K iterations. Table 8.5 collects the outcome of this experiment. We can notice how variants (A) and (B) trained on synthetic data are greatly improved by the fine-tuning, while (C) achieves slightly lower accuracy after fine-tuning. Despite allowing for much higher generalization to real images, the supervision allowed by our method is *weaker* than the one obtained through real image pairs and perfect ground-truth. Thus, it is not surprising that networks trained from scratch to the end on perfect ground-truth might yield better accuracy. Nonetheless, combining synthetic data with our depthstilled images (D) allows for the best performance, confirming the findings from our previous experiments that a combination of the two worlds – synthetic data with perfect labels and realistic yet imperfect images and labels – is beneficial.

	Pre-training	Fine-tuning	KITTI12		KITTI15	
			EPE	Fl	EPE	Fl
(A)	Ch	✗	5.14	34.64	15.56	47.29
	Ch	✓	1.42	4.86	2.40	8.49
(B)	Ch→Th	✗	2.40	10.49	9.04	25.53
	Ch→Th	✓	<u>1.36</u>	<u>4.67</u>	<u>2.22</u>	<u>8.09</u>
(C)	dCOCO	✗	1.82	6.62	5.09	16.72
	dCOCO	✓	1.37	4.70	2.76	9.15
(D)	Ch→Th→dCOCO	✗	1.78	7.00	4.82	18.03
	Ch→Th→dCOCO	✓	**1.32**	**4.54**	**2.21**	**7.93**

TABLE 8.5: **Comparison with synthetic datasets – fine-tuning.** Performance of RAFT variants pre-trained on synthetic datasets (A) and (B), on dCOCO (C) or both (D) when fine-tuned on a subset of 160 images from KITTI 2015, tested on KITTI 2012 and the remaining 40 images from KITTI 2015.

Impact on different optical flow networks. To prove that the superior generalization we achieve is enabled by our data rather than a specific architecture such as RAFT, we also train PWCNet [287] on the 20K images generated from COCO. Table 8.6 shows how PWCNet trained on dCOCO (C) dramatically outperforms the original variants trained on Chairs (A) and fine-tuned on Things (B) when testing on real data, at the cost of lower performance on Sintel synthetic images, substantially confirming our findings from previous experiments with RAFT, reported in the table for comparison (D). This fact proves that our data, generated from single yet realistic still images, significantly improves generalization to real data independently from the optical flow model trained.

Model	Dataset	Sintel C.		Sintel F.		KITTI12		KITTI15	
		EPE	> 3	EPE	> 3	EPE	Fl	EPE	Fl
(A) PWCNet	Ch	3.33	-	4.59	-	5.14	28.67	13.20	41.79
(B) PWCNet	Ch$\rightarrow$Th	**2.55**	-	**3.93**	-	4.14	21.38	10.35	33.67
(C) PWCNet	dCOCO	4.14	11.54	5.57	15.58	**3.16**	**13.30**	**8.49**	**26.06**
(D) RAFT	dCOCO	2.63	7.00	3.90	11.31	1.82	6.62	3.81	12.42

TABLE 8.6: **Impact of depthstillation on different architectures**. Evaluation on PWC-Net and RAFT. Entries with "-" are not provided in the original paper.

8.2.6 Comparison with self-supervision from videos

Given the rich literature about self-supervised optical flow [194, 169, 170, 130], we compare our strategy with state-of-the-art practises for self-supervised optical flow [130]. In contrast to most works in this field that train and test in the same domain [194, 169, 170, 130], we inquire about how well networks trained in a self-supervised manner or leveraging our proposal transfer across different real datasets. To this aim, we adopt DAVIS [223] for training and evaluate on KITTI 2012 and 2015 as in the previous experiments. To train UFlow [130], we use the official code provided by the authors. In particular, we trained the model on the entire DAVIS dataset for 1M steps, using a batch size of 1 as suggested in [130], 512×384 resized images and letting unchanged other configuration parameters in order to replicate the authors' settings. Being UFlow based on PWCNet, we train from scratch another instance of PWCNet on dDAVIS for the same number of steps with a batch of 8 over depthstilled images and labels. The learning rate scheduling is the same highlighted in section 8.2.3, while the crop is 512×384. This way, we evaluate how well a PWCNet trained on depthstilled data transfers to other datasets compared to a model trained on real videos framing the same image content of the depthstilled images. Table 8.7 collects the outcome of this experiment. We can notice how the PWCNet model trained on dDAVIS (B) transfers much better to the KITTI 2012 and 2015 datasets compared to UFlow trained on the real DAVIS (A), thanks to the stronger supervision from the distilled optical flow labels. For the sake of completeness, we also report the results achieved by RAFT (C) trained on the same data, confirming to be superior.

Model	Dataset	KITTI12		KITTI15	
		EPE	Fl	EPE	Fl
(A) UFlow	DAVIS	3.49	14.54	9.52	25.52
(B) PWCNet	dDAVIS	**2.81**	**11.29**	**6.88**	**21.87**
(C) RAFT	dDAVIS	1.78	6.85	3.80	13.22

TABLE 8.7: **Comparison between self-supervision and depthstillation – generalization.** Effectiveness of the two strategies when evaluated on unseen data (KITTI 2012 and 2015).

8.2.7 Limitations.

Our pipeline has some obvious limitations. Indeed, the training samples we generate are far from being utterly realistic because cannot model some behaviors, such as the large 3D rotation of objects in the scene, frequently found in real videos. Thus, despite the strong generalization we achieve compared to self-supervision, real videos allow for much better specialization when training and testing in the same domain. As shown in Table 8.8, UFlow trained on the 4K images of the KITTI multiview dataset (A) performs much better than PWCNet trained on 960×288 crops from dKITTI (B), a set of about 4K images depthstilled from KITTI 2015 multiview testing set. On the other hand, RAFT trained on dKITTI with the same crop size (C) gets closer to UFlow, thanks to the more effective architecture.

	Model	Dataset	KITTI12		KITTI15	
			EPE	Fl	EPE	Fl
(A)	UFlow	KITTI	-	-	**3.08**	**10.00**
(B)	PWCNet	dKITTI	2.64	9.43	7.92	22.17
(C)	RAFT	dKITTI	1.76	5.91	4.01	13.35

TABLE 8.8: **Comparison between self-supervision and depthstillation – specialization.** Effectiveness of the two strategies when training and testing on similar data (KITTI 2015). Entries with "-" are not provided in the original paper.

This lower specialization is also due to the completely random motions we depthstill. In contrast, KITTI motions consist of a much smaller subset (i.e. mostly forward translations or steerings) dominant in the real KITTI multiview split, yet rarely occurring in dKITTI.

As take-home message, our depthstillation strategy effectively addresses the scarcity of training data, e.g. when annotated images or not-annotated videos of the target environment are not available, yielding superior generalization compared to existing practices. Moreover, it is complementary to domain-specific real training data with labels, seldom ever available in practice.

8.3 Traditional vs learned inpainting

Figure 8.5 illustrates the effects of different inpainting strategies to fill holes in $\mathcal{I}_1$. Most recent inpainting methods are based on deep learning [183] and, of course, they require additional supervision that would add complexity to our pipeline, although not introducing sensible improvements on the quality of $\mathcal{I}_1$.

We show the outcome of some of these strategies, sorted by increasing complexity. From left to right, the results of 1) background texture filling [320], that consists of filling invalid pixels with RGB values taken from another image after color alignment, 2) the traditional image inpainting [295] used in our pipeline, 3) the predictions of a GAN model [183] pre-trained for image inpainting and 4) the prediction of a Fourier Features Network [290], trained directly on the image itself and thus optimized for each single $\mathcal{I}_1$. For the latter [290], we train a compact MLP to predict RGB values given as input the 2D coordinates, remapped into Fourier Features, of valid pixels in $\mathcal{I}_1$. Then, we use the trained model to infer the RGB for invalid pixels in $\mathcal{I}_1$. Notice that, in our pipeline, this setting would require a standalone training over each image in the dataset, dramatically increasing the complexity of our solution.

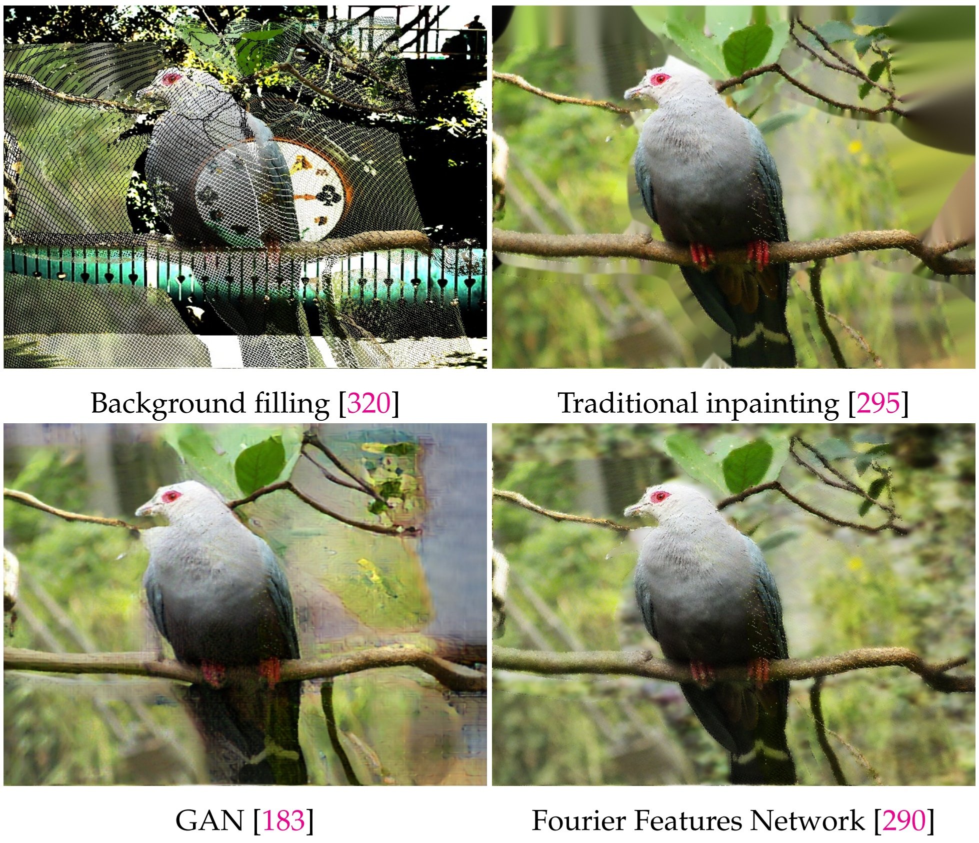


FIGURE 8.5: **Impact of different inpainting strategies.** We run different inpainting methods to fill holes in $\mathcal{I}_1$.

These qualitative examples highlight that background filling, although useful when generating stereo images [320], is not enough in the case of 2D motions. Moreover, large occluded regions result challenging to fill even for deep learning models [183]. The Fourier Features Network [290] results in more visually pleasant images, yet turns out to be prohibitive in terms of time required to depthstill thousand of images. Hence, in our pipeline, we rely on [295] since it provides comparable results with minimum complexity.

8.3.1 Qualitative examples

We show a detailed qualitative example from dCOCO in Figure 8.6. We first depict images generated without taking into account depth, i.e. assuming a constant distance from the camera enabling only planar motions and rotations. We can notice how this produces a meager variety of flow vectors on a single frame. Leveraging depth, we can model more complex flow fields, leading to much more variegate vectors occurring on the same scene. As shown in Table 8.1, optical flow networks trained on these images are dramatically more accurate. Finally, instance segmentation allows for extracting objects from the scene and simulate independent motions. This additional strategy results in even more variegate flow fields, although increasing the accuracy of optical flow networks marginally compared to the previous case.

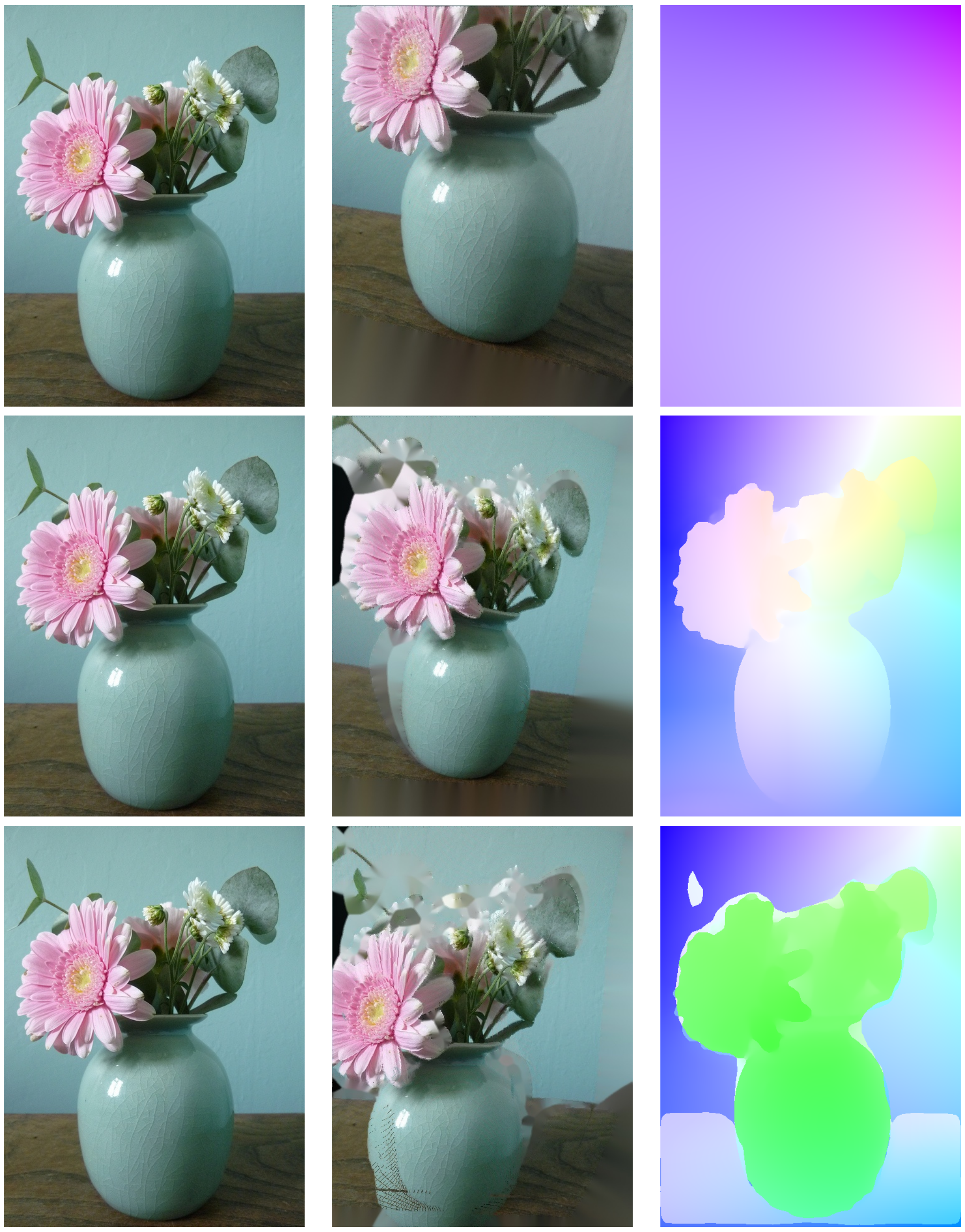


FIGURE 8.6: **Qualitative examples of dCOCO.** From left to right, $\mathcal{I}_0$, $\mathcal{I}_1$ and $\mathcal{F}_{0\rightarrow 1}$. First row depicts the result obtained by using a constant depth map, while second and third rows illustrate the final results when including also the monocular depth map and independent motions respectively.

8.4 Conclusions

In this chapter, a novel framework called depthstillation has been presented. With depthstillation we can generate, with a low-effort, a large amount of training data, i.e. novel views coupled with ground-truth labels, for optical flow network. The framework does not cost as a conventional supervised strategy, since human-intervention or other kind of sensors are not required at all, but in constant to unsupervised methods it provides a better and dense supervision for all the pixels in the image.

Chapter 9

Stereo depth estimation aided by monocular supervision

Chapter 4 shows that stereo labels, sourced by classical methods, are particularly effective in boosting the performance of monocular model. This strategy has been applied also in parallel works [321] with the same purpose. However, stereo and monocular exploit a different reasoning: on the one hand, stereo relies on geometry and tries to solve a matching problem, while monocular, on the other hand, is forced to look for other cues and priors, such as occlusions or shadows. Although the monocular case is generally more difficult to solve, stereo suffers especially when the matching fails, such as due to occlusions. Fortunately, these problems are non-overlapping, and the results of [302, 321] support this claim at least for the monocular case. In fact, in these works stereo labels lead to stronger supervision at training time than image reconstruction loss, allowing to escape from local minimums in challenging regions. On the other hand, monocular solutions do not have to solve the matching, thus do not suffer from the occlusion problem. Nonetheless, this evidence has not been leveraged adequately in the past literature. In this chapter we show how a peculiar monocular model is beneficial in boosting the performance of stereo networks. Starting from a stereo dataset without depth labels, we first train a monocular model leveraging classical stereo methods as additional supervision. Then, we use this expert model to generate labels aimed at training a stereo model on the same dataset. The monocular model takes as input not only the image but also a few labels sourced by classical methods. We exploit the effect due to this additional input to realize a filtering strategy aimed at removing the most unreliable regions in monocular maps, obtaining better supervision for the stereo model. The proposed training scheme is effective and leads to stereo models with good generalization capabilities and robustness against occlusions. This chapter is based on Reversing the cycle: self-supervised deep stereo through enhanced monocular distillation [11].

9.1 Method

This section describes the strategy that allows us to distill highly accurate disparity annotations for a stereo dataset made up of raw rectified images only and then use them to supervise deep stereo networks. It is worth noting that, by abuse of notation, we use depth and disparity interchangeably although our proxy extraction method works entirely in the disparity domain. For our purposes, we rely on two main stages, as depicted in Figure 9.1: 1) we train a monocular completion network (MCN) from sparse disparity points sourced by traditional stereo methods and 2)

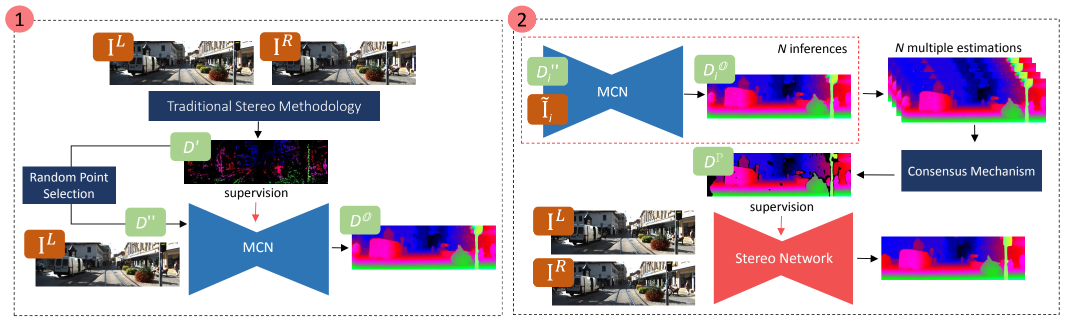


FIGURE 9.1: **Framework overview.** ① Sparse disparity points from a traditional stereo method are given as input to a monocular completion network (MCN). Then, in ② we leverage MCN to distill accurate proxies through the proposed consensus mechanism. Such labels guide the training of a deep stereo network.

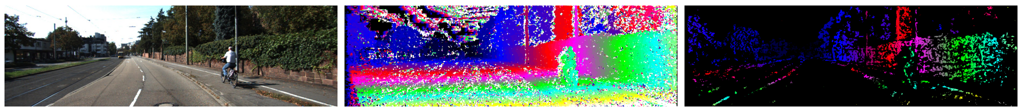


FIGURE 9.2: **Disparity map filtering.** From left to right, reference image from KITTI, the noisy disparity map computed by [351] and the outcome of filtering [305].

we train deep stereo networks using highly reliable points from MCN, selected by a novel consensus mechanism.

9.1.1 Monocular Completion Network (MCN)

Stereo algorithms struggle on occluded regions due to the difficulties to find correspondences between images. On the contrary, monocular methods do not rely on matching and thus, they are potentially not affected by this problem. In this stage, our goal is to obtain a strong guidance even on occluded areas relying on a monocular depth network. However, monocular estimates intrinsically suffer the scale factor ambiguity due to the lack of geometric constraints. Therefore, since stereo pairs are always available in our setup, we also leverage on reliable sparse disparity input points from traditional stereo algorithms in addition to the reference image. Thanks to this combination, MCN is able to predict dense depth maps preserving geometrical information.

Reliable disparity points extraction. At first, we rely on a traditional stereo matcher $\mathcal{S}$ (e.g. [351]) to obtain an initial disparity map D from a given stereo pair $(\mathcal{I}^L, \mathcal{I}^R)$ as

$$D = \mathcal{S}(\mathcal{I}^L, \mathcal{I}^R) \tag{9.1}$$

However, since such raw disparity map contains several outliers, especially on ill-posed regions such as occlusions or texture-less areas as it can be noticed in Figure 9.2, a filtering strategy $\mathcal{F}$ (e.g. [305]) is applied to discard spurious points

$$D' = \mathcal{F}(\mathcal{S}(\mathcal{I}^L, \mathcal{I}^R)) \tag{9.2}$$

By doing so, only a subset D' of highly reliable points is preserved from D at the cost of a sparser disparity map. However, most of them do not belong to occluded regions thus not enabling supervision on such areas. This can be clearly perceived observing the outcome of a filtering strategy in Figure 9.2.

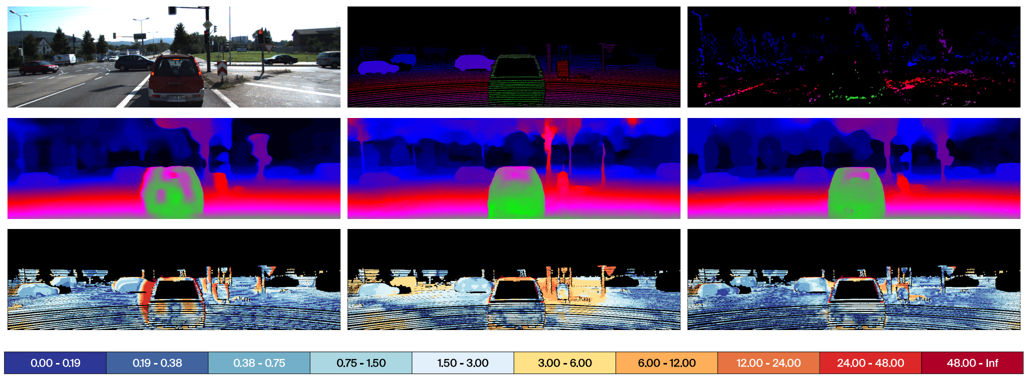


FIGURE 9.3: **Occlusion handling and scale recovery.** The first row depicts the reference image from KITTI, the ground-truth and the disparity map by [351] filtered with [305]. In the middle, from left to right the output of monocular depth network [302] trained without occlusion augmentation, the same network using the occlusion augmentation and our MCN. Last row shows the corresponding error maps. Best viewed with colors.

Monocular disparity completion. Given D', we deploy a monocular completion network, namely MCN, in order to obtain a dense map D^{O}. We self-supervise MCN from stereo and, as in [80], to handle occlusions we horizontally flip $(\mathcal{I}^L, \mathcal{I}^R)$ at training time with a certain probability without switching them. Consequently, occluded regions (e.g. the left border of objects) are randomly swapped with not-occluded areas (e.g. the right borders), preventing to always expect high error on left and low error on right borders, thus forcing the network to handle both. This strategy turns out ineffective in case of self-supervised stereo, since after horizontal flip the stereo pair have to be switched in order to keep the same search direction along the epipolar line, thus making occlusions occur in the same regions. Even if this technique helps to alleviate errors in occluded regions, a pure monocular network struggles compared to a stereo method at determining the correct depth. This is well-known in the literature and shown in our experiments as well. Thus, we adopt a completion approach leveraging sparse reliable points provided by a traditional stereo method constraining the predictions to be properly scaled. Given the set of filtered points, only a small subset D'', with $||D''|| \ll ||D'||$, is randomly selected and used as input, while D' itself is used for supervision purposes. The output of MCN is defined as

$$D^{O} = \text{MCN}(\mathcal{I}^L, D'' \xleftarrow{p \ll 1} (D')) \tag{9.3}$$

with $x \xleftarrow{p} (y)$ a random uniform sampling function extracting x values out of y per-pixel values with probability p. This sampling is crucial to both improve MCN accuracy, as shown in our experiments, as well as for the final distillation step discussed in the remainder. Once trained, MCN is able to infer scaled dense disparity maps D^{O}, as can be perceived in Figure 9.3. Looking at the rightmost and central disparity maps, we can notice how the augmentation protocol enables to alleviate occlusion artifacts. Moreover, our overall completion strategy, compared to the output of the monocular network without disparity seeds (leftmost and center disparity maps), achieves much higher accuracy as well as correctly handles occlusions. Therefore, we effectively combine stereo from non-occluded regions and monocular prediction in occluded areas. Finally, we point out that we aim at specializing MCN on the training set to generate labels on it since its purpose is limited to distillation.

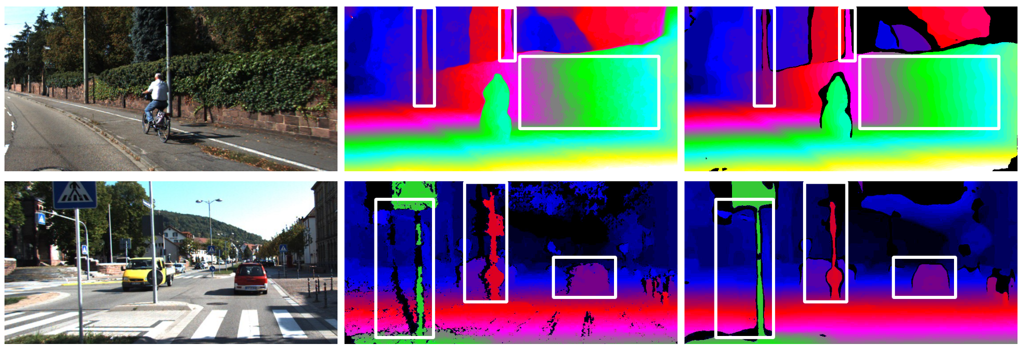


FIGURE 9.4: **Proxy distillation.** The first row depicts, from left to right, the reference image, the disparity map computed by a single inference of MCN and the one filtered and regularized using our consensus mechanism. The second row shows the reference image, the disparity map generated by SGM [101] filtered using the left-right consistency check strategy and our disparity map. Images from KITTI.

9.1.2 Proxy distillation for deep stereo

Eventually, we leverage the trained MCN to distill offline proxy labels beneficial to supervise stereo networks. However, such data might still contain some inconsistent predictions, as can be perceived in the rightmost disparity map of Figure 9.3. Therefore, our goal is to discard them, keeping trustworthy reliable depth estimates to train deep stereo networks.

Consensus mechanism and distillation. To this aim, given an RGB image $\mathcal{I}$ and the relative D', we perform N inferences of MCN by feeding it with D''_i and $\tilde{\mathcal{I}}_i$, with $i \in [1, N]$. Respectively, D''_i is sampled from D' according to the strategy introduced in Sec. 3.1 and $\tilde{\mathcal{I}}_i$ is obtained through random augmentation (explained later) applied to $\mathcal{I}$. This way, we exploit consistencies and contradictions among multiple $D^{\mathbb{O}}_i$ to obtain reliable proxy labels $D^{\mathbb{P}}$, defined as

$$D^{\mathbb{P}} \xleftarrow{\sigma^2(\{D^{\mathbb{O}}_i\}_{i=1}^N)<\gamma} \mu(\{D^{\mathbb{O}}_i\}_{i=1}^N) \tag{9.4}$$

where $x \xleftarrow{\sigma^2(y)<\gamma} \mu(y)$ is a function that, given N values y for the same pixel, samples the mean value $\mu(y)$ only if the variance $\sigma^2(y)$ is smaller γ. As distillation is performed offline, this step does not need to be differentiable.

Figure 9.4 shows that such a strategy allows us to largely regularize $D^{\mathbb{P}}$ compared to $D^{\mathbb{O}}$, preserving thin structures, e.g. the poles on the right side, yet achieving high density. It also infers significant portions of occluded regions compared to proxies sourced from traditional methods (e.g. SGM).

Deep stereo training. Once highly accurate proxy labels $D^{\mathbb{P}}$ are available on the same training set, we exploit them to train deep stereo networks in a weakly-supervised manner. In particular, a regression loss is used to minimize the difference between stereo predictions and $D^{\mathbb{P}}$.

9.2 Experiments

In this section, we thoroughly evaluate our proposal, proving that sourcing labels with a monocular completion approach is beneficial to train deep stereo networks.

9.2.1 Implementation details

Traditional stereo methods. We consider two main non-learning based solutions, characterized by different peculiarities, to generate accurate sparse disparity points from a rectified stereo pair. In particular, we use the popular semi-global matching algorithm SGM [101], exploiting the left-right consistency check (LRC) to remove wrong disparity assignments, and the WILD strategy proposed in [305] that selects highly reliable values from the maps computed by the local algorithm Block-Matching (BM) [351] exploiting traditional confidence measures. We refer to these methods (i.e. stereo method followed by a filtering strategy) as SGM/L and BM/W, respectively.

Monocular Completion Network. We adopt the weakly-supervised monocular architecture MonoResMatch [302], discussed in Chapter 4, and we train it with the supervision of disparity proxy labels specifically suited for our purposes. We modify the network to exploit accurate sparse annotations as input by concatenating them with the RGB image. We set the random sampling probability in Eq. 9.3 as $p = \frac{1}{1000}$. In our experiments, we train from scratch the MCN network following the same training protocol defined in [302] except for the augmentation procedure which includes the flipping strategy (with 0.25 probability) aimed at handling occlusion artifacts [80]. Instead, we empirically found out that generating $D^{\mathbb{O}}$ using a larger set of points helps to achieve more accurate predictions at inference time. In particular, we fix $p = \frac{1}{20}$ and $p = \frac{1}{200}$ for BM/W and SGM/L respectively. Finally, for the consensus mechanism, we fix $N = 50$, the threshold $\gamma = 3$ and apply for each $\mathcal{I}_i$ color augmentation and random horizontal flip (with 0.5 probability).

Stereo networks. We considered both 2D and 3D deep stereo architectures, ensuring a comprehensive validation of our proposal. In particular, we designed a baseline architecture, namely Stereodepth, by extending [80] to process stacked left and right images, and iResNet [165] as examples of the former case, while PSMNet [43] and GWCNet [90] as 3D architectures. At training time, the models predict disparities D^S at multiple scales in which each intermediate prediction is upsampled at the input resolution. A weighted smooth L1 loss function (the lower the scale, the lower the weight) minimizes the difference between D^S and the disparity provided by the proxy $D^{\mathbb{P}}$ considering only valid pixels, using Adam [139] as optimizer ($\beta_1 = 0.9$ and $\beta_2 = 0.999$). We adopt the original PyTorch implementation of the networks if available. Moreover, all the models have been trained to fit a single Titan X GPU.

9.2.2 Evaluation of proxy label generators

At first, we first evaluate the accuracy of proxies produced by our approach with respect to traditional methods. We consider both D1 and EPE, computed on disparities, as error metrics on both non-occluded (*Noc*) and all regions (*All*). These metrics have been already introduced in Chapter 3.2. In addition, the density and the overlap with the ground-truth are reported to take into account filtering strategies. Table 9.1 reports a thorough evaluation of different methodologies and filtering techniques. It can be noticed how BM and SGM have different performances due to their complementarity (local vs semi-global), but containing several errors. Filtering strategies help to remove outliers, at the cost of sparser maps. Notice that restoring the full density through *hole-filling* [101] slightly improves the results of SGM/L, but it is not meaningful for BM/W since filtered maps are too sparse. Unsurprisingly, even if the depth maps produced by the vanilla MonoResMatch are fully-dense, they

Method	Configuration					Statistics		Noc		All	
	Source	Filter	A	R	C	Density(%)	Overlap(%)	D1	EPE	D1	EPE
MONO	MonoResMatch	-	-	-	-	100.0	100.0	26.63	2.96	27.00	2.99
BM	BM	-	-	-	-	100.0	100.0	34.48	16.14	35.46	16.41
SGM	SGM	-	-	-	-	100.0	100.0	6.65	1.67	8.12	2.16
BM/L	BM	LRC	-	-	-	57.89	62.09	16.09	6.42	16.22	6.46
SGM/L	SGM	LRC	-	-	-	86.47	92.28	3.99	1.00	4.01	1.00
SGM/L(*hole-filling*)	SGM	LRC	-	-	-	100.0	100.0	6.56	1.34	7.68	1.57
BM/W	BM	WILD	-	-	-	12.33	10.43	1.33	0.81	1.35	0.81
MCN-SGM/L	SGM	LRC	-	-	-	100.0	100.0	6.36	1.27	7.80	1.50
MCN-SGM/L-R	SGM	LRC	-	✓	-	100.0	100.0	5.28	1.13	5.73	1.21
MCN-SGM/L-AC	SGM	LRC	✓	-	✓	95.36	97.36	5.58	1.17	5.58	1.15
MCN-SGM/L-RC	SGM	LRC	-	✓	✓	93.50	96.32	2.95	0.86	3.14	0.89
MCN-SGM/L-ARC	SGM	LRC	✓	✓	✓	92.53	95.76	2.78	0.84	2.92	0.86
MCN-BM/W	BM	WILD	-	-	-	100.0	100.0	11.86	1.93	12.50	2.03
MCN-BM/W-R	BM	WILD	-	✓	-	100.0	100.0	6.79	1.40	7.11	1.45
MCN-BM/W-AC	BM	WILD	✓	-	✓	91.45	94.76	8.36	1.53	8.64	1.57
MCN-BM/W-RC	BM	WILD	-	✓	✓	91.12	95.28	3.79	0.95	4.03	1.0
MCN-BM/W-ARC	BM	WILD	✓	✓	✓	86.82	93.56	3.16	0.90	3.27	0.92

TABLE 9.1: **Evaluation of proxy generators.** We tested proxies generated by different strategies on the KITTI 2015 training set. A means RGB augmentation to input images, R means random selection of input points while C indicates the consensus strategy.

are not accurate due to its inherent monocular nature. On the contrary, our monocular strategy MCN produces dense yet accurate maps thanks to the initial disparity guesses, regardless the sourcing stereo algorithm. Moreover, by applying Augmentation techniques (A) on the RGB image or selecting Random input points (R), allow to increase variance and to exploit our Consensus mechanism (C) to filter out unreliable values, thus achieving even better results. In fact, the consensus mechanism is able to discard wrong predictions preserving high density, reaching best performances when A and R are both applied. It is worth noting that if R is not performed, the network is fed with all the available guesses both at training and testing time, with remarkably worse results compared to configuration using random sampling.

Disparity completion comparison. We validate our MCN combined with the consensus mechanism comparing it to GuideNet [338], a supervised architecture designed to generate high-quality disparity annotations exploiting multi-frame LiDAR points and stereo pairs as input. Following [338], we measure the valid pixels, correct pixels, and accuracy (i.e. 100.0 - D1) on 142 images of the KITTI 2015 training set. Table 9.2 clearly shows how MCN trained in a weakly-supervised manner achieves comparable accuracy with respect to GuideNet-LiDAR by exploiting sparse disparity estimates from both [305] and [101] but with a significantly higher number of points, even on foreground regions (Obj). Notice that LiDAR indicates that the network is fed with LiDAR points filtered according to [308]. To further demonstrate the generalization capability of MCN to produce highly accurate proxies relying on points from heterogeneous sources, we feed MCN-BM/W-R with raw LiDAR measurements. By doing so, our network notably outperforms GuideNet in this configuration, despite it leverages a single RGB image and has not been trained on LiDAR points.

Model	All			Obj		
	Valid	Correct	Accuracy (%)	Valid	Correct	Accuracy (%)
MCN-BM/W-ARC	11,551,461	11,247,966	97.37	1,718,267	1,642,872	95.61
MCN-SGM/L-ARC	12,201,763	11,860,923	97.20	1,788,154	1,672,222	93.52
MCN-LiDAR	11,773,897	11,636,787	98.83	1,507,222	1,459,726	96.84
GuideNet-LiDAR [338] †	2,973,882	2,915,110	98.02	221,828	210,912	95.07

TABLE 9.2: **Model-guided comparison**. Comparison between our MCN model and the supervised GuideNet stereo architecture [338] using 142 ground-truth images of the KITTI 2015 training set. † indicates that the network requires LiDAR points at training time. Accuracy is defined as 100-D1.

9.2.3 Ablation study

In this subsection, we support the statement that a completion approach provides a better supervision compared to traditional stereo algorithms. We first run experiments on KITTI and then use our best configuration on DrivingStereo as well, showing that it is effective on multiple large stereo datasets.

KITTI. For the ablation study, reported in Table 9.3, we consider both 3D (PSMNet) and 2D (Stereodepth) networks featuring different computational complexity. First, we train the baseline configuration of the networks, i.e. relying image reconstruction loss functions (PHOTO) only as in [80]. Then, we leverage disparity values sourced by traditional stereo algorithms in which outliers have been removed by the filtering strategies adopted. Such labels provide a useful guidance for stereo networks and allow to obtain more accurate models than the baselines. Nonetheless, proxies produced by MCN prove to be much more effective than traditional ones, improving both D1 and EPE by a notable margin regardless the stereo algorithm used to extract the input guesses. Moreover, it can be perceived that best results are obtained when the complete consensus mechanism is enabled.

Backbone	Supervision	Noc		All	
		D1	EPE	D1	EPE
Stereodepth	PHOTO	**6.50**	**1.30**	**7.12**	**1.40**
PSMNet	PHOTO	6.62	**1.30**	7.67	1.50
Stereodepth	SGM-L	5.22	**1.13**	5.43	**1.15**
Stereodepth	SGM/L(*hole-filling*)	6.06	1.16	6.38	1.21
Stereodepth	BM/W	**5.19**	1.16	**5.37**	1.18
PSMNet	SGM/L	5.46	1.19	5.61	1.21
PSMNet	SGM/L(*hole-filling*)	6.06	1.23	6.32	1.26
PSMNet	BM/W	6.89	1.59	7.03	1.60
Stereodepth	MCN-SGM/L-R	5.11	1.11	5.37	1.14
Stereodepth	MCN-BM/W-R	4.75	1.05	4.96	1.07
Stereodepth	MCN-SGM/L-ARC	4.56	1.08	4.77	1.11
Stereodepth	MCN-BM/W-ARC	4.21	1.06	4.39	1.07
PSMNet	MCN-SGM/L-R	4.39	1.05	4.60	1.07
PSMNet	MCN-BM/W-R	4.30	1.06	4.49	1.08
PSMNet	MCN-SGM/L-ARC	4.02	1.05	4.20	1.07
PSMNet	MCN-BM/W-ARC	<u>**3.68**</u>	<u>**0.99**</u>	<u>**3.85**</u>	<u>**1.01**</u>
Stereodepth	LiDAR/SGM [308]	3.95	1.07	4.10	1.09
PSMNet	LiDAR/SGM [308]	**3.93**	**1.05**	**4.07**	**1.07**

TABLE 9.3: **Ablation study.** We trained Stereodepth and PSMNet [43] on KITTI using supervision signals from different proxy generators and tested on KITTI 2015.

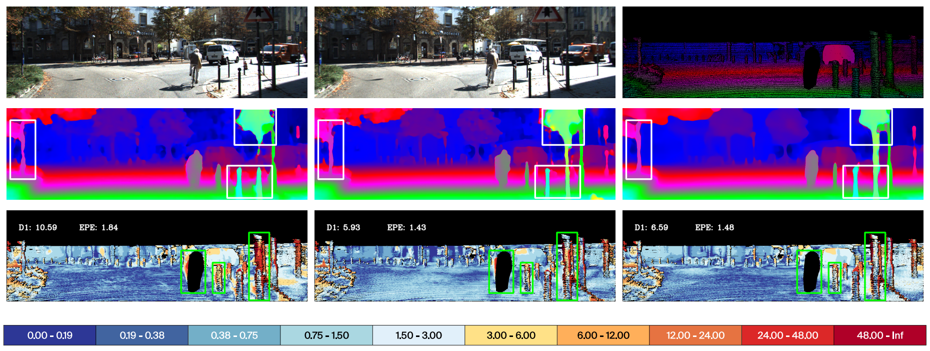


FIGURE 9.5: **Impact of proxies.** From top, input stereo pair and ground-truth disparity map, predictions by Stereodepth trained with SGM/L (left), LiDAR (center) and our MCN-BM/W-ARC (right), error maps. Best viewed with colors.

Finally, we rely also on filtered LiDAR measurements from [308] in order to show differences with respect to supervision from active sensors. Noteworthy, models trained using proxies distilled by ARC configuration of MCN prove to be comparable or even better than using LiDAR with PSMNet and Stereodepth. This behaviour can be explained due to a more representative and accurate supervision on occluded areas than traditional stereo and filtered LiDAR, thus making the deep networks more robust even there, as clearly shown in Figure 9.5.

DrivingStereo. We validate the proposed strategy also on DrivingStereo, proving that our distillation approach is able to largely improve the performances of stereo networks also on different datasets. In particular, in Table 9.4 we compare Stereodepth and PSMNet errors when trained using MCN-BM/W-ARC method (i.e. the best configuration on KITTI) with LiDAR and BM/W. Again, our proposal outperforms BM/W, and reduces the gap with high quality LiDAR supervision. Moreover, to verify generalization capabilities, we test on KITTI also correspondent models trained on DrivingStereo, without performing any fine-tuning (DS $\rightarrow$ K), and vice versa (K $\rightarrow$ DS). It can be noticed that the gap between KITTI models (see Table 9.3) and those trained on DrivingStereo gets smaller, proving that the networks are able to perform matching correctly even in cross-validation scenario. We want to point out that this is due to our proxies, as can be clearly perceived by looking at rows 1-2 vs 3-4 in Table 9.4.

		Source $\rightarrow$ Target					
Backbone	Supervision	DS $\rightarrow$ DS		K $\rightarrow$ DS		DS $\rightarrow$ K	
		D1	EPE	D1	EPE	D1	EPE
Stereodepth	BM/W	**4.46**	**1.20**	**4.67**	**1.10**	**6.35**	**1.36**
PSMNet	BM/W	8.81	1.94	5.06	1.30	7.07	1.65
Stereodepth	MCN-BM/W-ARC	2.47	0.94	2.97	0.96	5.64	1.22
PSMNet	MCN-BM/W-ARC	**1.87**	**0.86**	**2.32**	**0.88**	**5.16**	**1.17**
Stereodepth	LiDAR [338]	1.20	0.69	3.60	1.23	4.57	**1.17**
PSMNet	LiDAR [338]	**0.59**	**0.54**	**2.64**	**1.03**	**4.52**	1.26

TABLE 9.4: **Cross-validation analysis.** We tested on the Target dataset models trained on the Source one, leveraging different proxies. Notice that no fine-tuning on the target dataset is performed in case of cross-validation.

9.2.4 Comparison with state-of-the-art

We compare our models with state-of-the-art self-supervised stereo methods. Table 9.5 reports, in addition to D1 and EPE, also RMSE and RMSE log as depth error measurements and $\delta < 1.25, \delta < 1.25^2, \delta < 1.25^3$ accuracy metrics according to [318, 363]. Notice that some of these methods exploit additional information, such as stereo videos [318] or adaptation strategies [361]. Proxies distilled by MCN-BM/W-ARC can be successfully exploited using both 2D and 3D architectures, enabling even the simplest 2D network Stereodepth to outperform all the competitors. Our strategy is effective, allowing all the adopted backbones to improve depth estimation by a notable margin on 6 metrics out of 7.

Method	RMSE	log RMSE	D1	EPE	$\delta <1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Godard et al.[79] (stereo)	5.742	0.202	10.80	-	0.928	0.966	0.980
Lai et al.[151]	4.186	0.157	8.62	1.46	0.950	0.979	0.990
Wang et al.[318] (stereo only)	4.187	0.135	7.07	-	0.955	0.981	0.990
Zhong et al.[362]	4.857	0.165	6.42	-	0.956	0.976	0.985
Wang et al.[318] (stereo videos)	3.404	0.121	5.94	-	0.965	0.984	0.992
Zhong et al.[361]*	**(3.176)**	(0.125)	(5.14)	-	(0.967)	-	-
Ours (Stereodepth)	3.882	0.117	4.39	1.07	0.971	0.988	**0.993**
Ours (GWCNet)	3.614	0.111	3.93	1.04	0.974	**0.989**	**0.993**
Ours (iResNet)	3.464	**0.108**	3.88	1.02	**0.975**	0.988	**0.993**
Ours (PSMNet)	3.764	0.115	**3.85**	**1.01**	0.974	0.988	**0.993**

TABLE 9.5: **Comparison with state-of-the-art**. Results of different self/weakly-supervised stereo networks on the KITTI 2015 training set with max depth set to 80m. Ours indicates networks trained using MCN-BM/W-ARC labels. * indicates networks trained on the same KITTI 2015 data, therefore not directly comparable with other methods.

Furthermore, we test our PSMNet trained using MCN-BM/W-ARC proxies on the KITTI 2015 online benchmark, reporting the results in Table 9.6. Our model not only outperforms [101] and self-supervised competitors, as can be also perceived in Figure 9.6, but also supervised strategies [301, 191] on both non-occluded and all areas.

Models	Dataset	E2E	D1-bg	D1-fg	D1-All	D1-Noc
Zbontar and LeCun (acrt) [353]	K	-	2.89	8.88	3.89	3.33
Tonioni et al. [301]	SF+K	✓	3.75	9.20	4.66	4.27
Mayer et al. [191]	SF+K	✓	4.32	4.41	4.34	4.05
Chang and Chen [43] (PSMNet)	SF+K	✓	1.86	4.62	2.32	2.14
Guo et al. [90] (GWCNet)	SF+K	✓	1.74	3.93	2.11	1.92
Zhang et al. [354]	SF+K	✓	**1.48**	**3.46**	**1.81**	**1.63**
Hirschmuller [101]	-	-	8.92	20.59	10.86	9.47
Zhou et al. [362]	K	✓	-	-	9.91	-
Li and Yuan [156]	K	✓	6.89	19.42	8.98	7.39
Tulyakov et al. [307]	K	-	3.78	10.93	4.97	4.11
Joung et al. [131]	K	-	-	-	4.47	-
Ours (PSMNet)	K	✓	**3.13**	**8.70**	**4.06**	**3.86**

TABLE 9.6: **KITTI 2015 online benchmark**. We submitted PSMNet, trained on MCN-BM/W-ARC labels, on the KITTI 2015 online stereo benchmark. In blue self/weakly-supervised methods, while in red fully-supervised strategies. We indicate with E2E architectures trained in an end-to-end manner, while SF on the SceneFlow dataset [191]

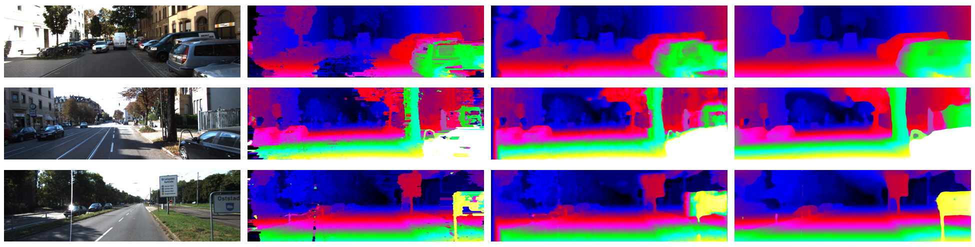


FIGURE 9.6: **KITTI 2015 online benchmark qualitatives**. From left to right, the reference images, and the disparity maps computed by [101], [156] and our PSMNet trained on MCN-BM/W-ARC labels.

9.2.5 Generalization

Finally, we show experiments supporting that supervision from our MCN-BM/W-ARC labels achieves good generalization to different domains. To this aim, we run our KITTI networks on Middlebury v3 and ETH3D, framing completely different environments.

Method	Training	Middlebury v3 [263]		ETH3D [269]	
	Dataset	BAD2	EPE	BAD2	EPE
Zhang et al.[354]	SF+K	**<u>18.90</u>**	**3.44**	**<u>3.43</u>**	**0.91**
Chang and Chen [43] (PSMNet)	SF+K	20.04	3.01	13.07	1.35
Guo et al.[90](GWCNet)	SF+K	21.36	3.29	19.96	1.88
Wang et al.[318](stereo only)	K	30.55	4.77	11.17	1.47
Wang et al.[318](stereo videos)	K	31.63	5.23	19.59	1.97
Lai et al.[151](stereo videos)	K	45.18	6.42	10.15	1.01
Ours (Stereodepth)	K	27.43	3.72	6.94	1.31
Ours (iResNet)	K	25.08	3.85	6.29	0.81
Ours(GWCNet)	K	20.75	3.17	**3.50**	**<u>0.48</u>**
Ours (PSMNet)	K	**19.56**	**<u>2.99</u>**	4.00	0.51

TABLE 9.7: **Generalization test on Middlebury v3 and ETH3D.** We evaluate networks trained in self/weakly-supervised (blue) or supervised (red) fashion on KITTI (K) and SceneFlow dataset (SF) [191]

Table 9.7 shows the outcome of this evaluation. We report, on top, the performance of fully supervised methods trained on SceneFlow [191] and fine-tuned on KITTI for comparison. On bottom, we report self/weakly-supervised frameworks trained on the KITTI split from the previous experiments. All networks are transferred without fine-tuning. Compared to existing self-supervised strategies (rows 4-6), networks trained with our proxies achieve much better generalization on both the datasets, performing comparable (or even better) with ground-truth supervised networks. Figure 9.7 shows few examples from the two datasets, where the structure of the scene is much better recovered when trained on our proxies.

9.3 Conclusions

This chapter proposed a novel framework to train stereo matching models when ground-truth labels are not available. Starting from an unlabeled stereo dataset, made of raw RGB pairs, we first obtain a peculiar monocular network aided by few

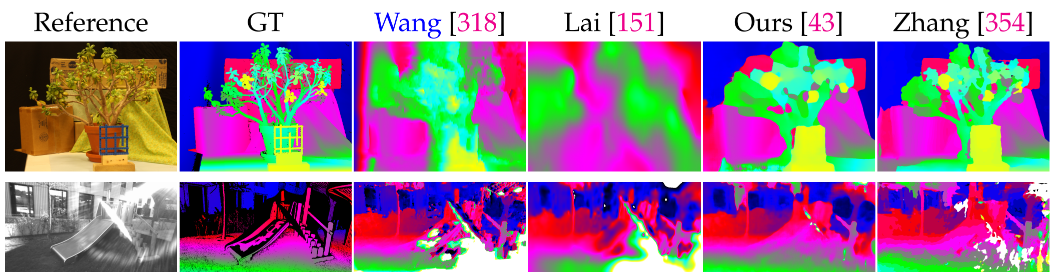


FIGURE 9.7: **Examples of generalization**. First row shows disparity maps obtained on a stereo pair from the Middlebury v3 dataset, while second from ETH3D. Zhang et al. [354] has been trained in supervised fashion, while the remaining methods are self/weakly-supervised.

stereo hints extracted by classical stereo methods. Then, we rely on this teacher network to train a student stereo network, exploiting a new consensus mechanism designed to filter out inconsistencies in monocular predictions. At the end of the training, the stereo network is ready to be used in applications and proves, thanks to the proposed strategy, to be quite robust even in hard-to-solve regions as near occlusions.

Chapter 10

Monocular depth uncertainty

In the previous chapters we have presented and discussed methods for obtaining depth values of the observed scene. Nonetheless, these observations in general do not have the same reliability everywhere: for example, for optical flow strategies fast motions of small objects are much harder to predict than small displacements of large entities, while ambiguous regions, such as texture-less or occluded areas, may lead to wrong depth and flow estimates. As pointed out in Chapter 2, confidence methods have been investigated for a long time in the past, especially for stereo and optical flow tasks. Conversely, the recent spread of monocular strategies has not gone hand in hand with methods aimed at evaluating their uncertainty. We now present a comprehensive evaluation of the uncertainty estimation in the monocular case, with particular interest in understanding how the self-supervised training paradigm generally adopted in monocular frameworks impacts uncertainty. The content of this chapter is based on the paper On the uncertainty of self-supervised monocular depth estimation [229].

10.1 Depth-from-mono and uncertainty

In this section, we introduce how to tackle uncertainty modelling with self-supervised depth estimation frameworks. Given a still image $\mathcal{I}$ any depth-from-mono framework produces an output map D encoding the depth of the observed scene. When full supervision is available, to train such a network we aim at minimizing a loss signal $\mathcal{L}_{fs}$ obtained through a generic function $\mathcal{F}$ of inputs estimated D and ground-truth D^* depth maps.

$$\mathcal{L}_{fs} = \mathcal{F}(D, D^*) \tag{10.1}$$

When traditional supervision is not available, it can be replaced by self-supervision obtained through image reconstruction. In this case, the ground-truth map D^* is replaced by a second image $\mathcal{I}^\dagger$. Then, by knowing camera intrinsics K, $K^\dagger$ and the relative camera pose $(R|t)$ between the two images, a reconstructed image $\tilde{\mathcal{I}}$ is obtained as a function π of intrinsics, camera pose, image $\mathcal{I}^\dagger$ and depth D, enabling to compute a loss signal $\mathcal{L}_{ss}$ as a generic $\mathcal{F}$ of inputs $\tilde{\mathcal{I}}$ and $\mathcal{I}$.

$$\mathcal{L}_{ss} = \mathcal{F}(\tilde{\mathcal{I}}, \mathcal{I}) = \mathcal{F}(\pi(\mathcal{I}^\dagger, K^\dagger, R|t, K, D), \mathcal{I}) \tag{10.2}$$

$\mathcal{I}$ and $\mathcal{I}^\dagger$ can be acquired either by means of a single moving camera or with a stereo rig. In this latter case, $(R|t)$ is known beforehand thanks to the stereo calibration parameters, while for images acquired by a single camera it is usually learned jointly to depth, both up to a scale factor. A popular choice for $\mathcal{F}$ is the weighted

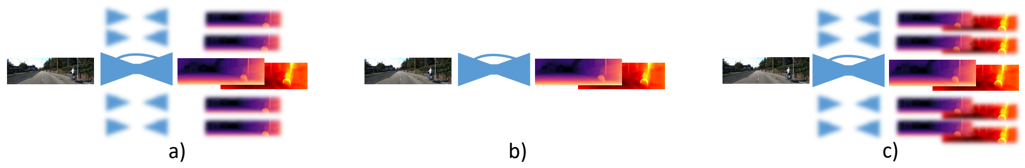


FIGURE 10.1: **Overview of uncertainty estimation implementations.** Respectively a) empirical methods model uncertainty as the variance of predictions from a subset of all the possible instances of the same network, b) predictive are trained to estimate depth and uncertainty as mean and variance of a distribution and c) Bayesian methods are approximated [207] by sampling multiple predictive models and summing single uncertainties with the variance of the depth predictions.

sum between $L1$ and Structured Similarity Index Measure (SSIM) [319] reported in equation 4.4.

In case of K frames used for supervision, coming for example by joint monocular and stereo supervision, for each pixel q the minimum among computed losses allows for robust reprojection [80]

$$\mathcal{L}_{ss}(q) = \min_{i \in [0..K]} \mathcal{F}(\tilde{\mathcal{I}}_i(q), \mathcal{I}(q)) \tag{10.3}$$

Traditional networks are deterministic, producing a single output typically corresponding to the mean value of the distribution of all possible outputs $p(D^*|\mathcal{I}, \mathcal{D})$, $\mathcal{D}$ being a dataset of images and corresponding depth maps. Estimating the variance of such distribution allows for modelling uncertainty on the network outputs, as shown in [120, 135] and depicted in Figure 10.1, a) in empirical way, b) by learning a predictive model or c) combining the two approaches.

First and foremost, we point out that the self-supervision provided to the network is *indirect* with respect to its main task. This means that the network estimates are not optimized with respect to the desired statistical distribution, i.e. depth D^*, but they are an input parameter of a function (π) optimized over a different statistical model, i.e. image $\mathcal{I}$. While this does not represent an issue for empirical methods, predictive methods like negative log-likelihood minimization can be adapted to this paradigm as done by Klodt and Vedaldi [143]. Nevertheless, we will show how this solution is sub-optimal when the camera pose is unknown, i.e. when π is function of two unknown parameters.

10.1.1 Uncertainty by image flipping

A simple strategy to estimate uncertainty is inspired by the post-processing (*Post*) step proposed by Godard et al.[79]. Such a refinement consists of estimating two depth maps D and $\overleftarrow{D}$ for image $\mathcal{I}$ and its horizontally flipped counterpart $\overleftarrow{\mathcal{I}}$. The refined depth map D^r is obtained by averaging D and $\overset{\rightleftarrows}{D}$, i.e. back-flipped $\overleftarrow{D}$. We encode the uncertainty for D^r as the difference between the two

$$u_{Post} = |D - \overset{\rightleftarrows}{D}| \tag{10.4}$$

that is, the variance over a small distribution of outputs (i.e., two), as typically done for empirical methods outlined in the next section. Although this method requires $2\times$ forwards at test time compared to the raw depth-from-mono model, as shown in Figure 10.2, it can be applied seamlessly to any pre-trained framework without any modification.

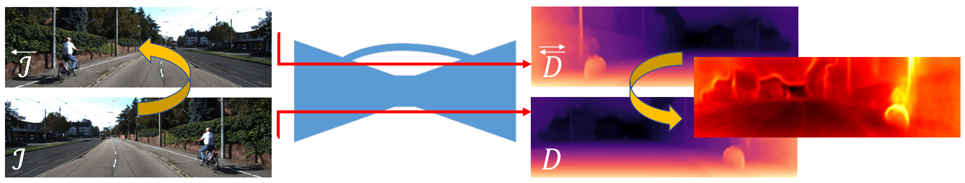


FIGURE 10.2: **Uncertainty by image flipping.** The difference between the depth D, inferred from image $\mathcal{I}$, and the depth $\overleftrightarrow{D}$, from the flipped image $\overleftarrow{\mathcal{I}}$, provides a basic form of uncertainty.

10.1.2 Empirical estimation

This class of methods aims at encoding uncertainty empirically, for instance, by measuring the variance between a set of all the possible network configurations. It allows to explain the model uncertainty, namely *epistemic* [135]. Strategies belonging to this category [119] can be applied to self-supervised frameworks straightforwardly.

Dropout Sampling (*Drop*). Early works estimated uncertainty in neural networks [186] by sampling multiple networks from the distribution of weights of a single architecture. Monte Carlo Dropout [281] represents a popular method to sample N independent models without requiring multiple and independent trainings. At training time, connections between layers are randomly dropped with a probability p to avoid overfitting. At test time, all connections are kept. By keeping dropout enabled at test time, we can perform multiple forwards sampling a different network every time. Empirical mean $\mu(D)$ and variance $\sigma^2(D)$ are computed, as follows, performing multiple (N) inferences:

$$\mu(D) = \frac{1}{N}\sum_{i=1}^{N} D_i \tag{10.5}$$

$$u_{Drop} = \sigma^2(D) = \frac{1}{N}\sum_{i=1}^{N}(D_i - \mu(D))^2 \tag{10.6}$$

At test time, using the same number of network parameters, N$\times$ forwards are required.

Bootstrapped Ensemble (*Boot*). A simple, yet effective alternative to weights sampling is represented by training an ensemble of N neural networks [153] randomly initializing N instances of the same architecture and training them with bootstrapping, i.e. on random subsets of the entire training set. This strategy produces N specialized models. Then, similarly to dropout sampling, we can obtain empirical mean $\mu(D)$ and variance $\sigma^2(D)$ in order to approximate the mean and variance of the distribution of depth values. It requires N$\times$ parameters to be stored, results of N$\times$ independent trainings, and a single forward pass for each stored configuration at test time.

Snapshot Ensemble (*Snap*). Although the previous method is compelling, obtaining ensembles of neural networks is expensive since it requires carrying out N independent training. An alternative solution [112] consists of obtaining N snapshots out of a single training by leveraging on cyclic learning rate schedules to obtain C pre-converged models. Assuming an initial learning rate λ_0, we obtain λ_t at any training iteration t as a function of the total number of steps T and cycles C as in [112]

$$\lambda_t = \frac{\lambda_0}{2} \cdot \left(\cos \left(\frac{\pi \cdot \mod (t-1, \lceil \frac{T}{C} \rceil)}{\lceil \frac{T}{C} \rceil} \right) + 1 \right) \tag{10.7}$$

Similarly to *Boot* and *Drop*, we obtain empirical mean $\mu(D)$ and variance $\sigma^2(D)$ by choosing N out of the C models obtained from a single training procedure.

10.1.3 Predictive estimation

This category aims at encoding uncertainty by learning a predictive model. This means that at test time these methods produce uncertainty estimates that are function of network parameters and the input image and thus reason about the current observations, modelling *aleatoric heteroscedastic* uncertainty [135]. Since often learned from real data distribution, for instance as a function of the distance between the predictions and the ground-truth or by maximizing log-likelihood, these approaches need to be rethought to deal with self-supervised paradigms.

Learned Reprojection (*Repr*). To learn a function over the prediction error employing a classifier is a popular technique used for both stereo [238, 273] and optical flow [185]. However, given the absence of ground-truth labels, we cannot apply this approach to self-supervised frameworks seamlessly. Nevertheless, we can drive one output of our network to mimic the behavior of the self-supervised loss function used to train it, thus learning ambiguities affecting the paradigm itself (e.g., occlusions, low texture and more). Indeed, the per-pixel loss signal is supposed to be high when the estimated depth is wrong. Thus, uncertainty u_{Repr} is trained adding the following term to $\mathcal{L}_{ss}$:

$$\mathcal{L}_{Repr} = \beta \cdot |u_{Repr} - \mathcal{F}(\tilde{\mathcal{I}}, \mathcal{I})| \tag{10.8}$$

Since multiple images $\mathcal{I}^\dagger$ may be used for supervision, i.e. when combining monocular and stereo [80], usually for each pixel q the minimum reprojection signal is considered to train the network, thus u_{Repr} is trained accordingly

$$\mathcal{L}_{Repr}(q) = \beta \cdot |u_{Repr}(q) - \min_{i \in [0..K]} \mathcal{F}(\tilde{\mathcal{I}}_i(q), \mathcal{I}(q))| \tag{10.9}$$

In our experiments, we set β to 0.1 and stop $\mathcal{F}$ gradients inside $\mathcal{L}_{Repr}$ for numerical stability. A similar technique appeared in [51], although not evaluated quantitatively.

Log-Likelihood Maximization (*Log*). Another popular strategy [212] consists of training the network to infer mean and variance of the distribution $p(D^*|\mathcal{I}, \mathcal{D})$ of parameters Θ. The network is trained by log-likelihood maximization (i.e., negative log-likelihood minimization)

$$\log p(D^*|w) = \frac{1}{N} \sum_q \log p(D^*(q)|\Theta(\mathcal{I}, w)) \tag{10.10}$$

w being the network weights. As shown in [119], the predictive distribution can be modelled as Laplacian or Gaussian respectively in case of $L1$ or $L2$ loss computation with respect to D^*. In the former case, this means minimizing the following loss function:

$$\mathcal{L}_{Log} = \frac{|\mu(D) - D^*|}{\sigma(D)} + \log \sigma(D) \tag{10.11}$$

with $\mu(D)$ and $\sigma(D)$ outputs of the network encoding mean and variance of the distribution. The additional logarithmic term discourages infinite predictions for any pixel. Regarding numerical stability [135], the network is trained to estimate the log-variance in order to avoid zero values of the variance. As shown by Klodt and Vedaldi [143], in absence of ground-truth D^* one can model the uncertainty u_{Log} according to photometric matching

$$\mathcal{L}_{Log} = \frac{\min_{i\in[0..K]} \mathcal{F}(\tilde{\mathcal{I}}_i(q), \mathcal{I}(q))}{u_{Log}} + \log u_{Log} \tag{10.12}$$

Recall that $\mathcal{F}$ is computed over π according to Equation 10.2. Although for stereo supervision this formulation is equivalent to traditional supervision, i.e. π is function of a single unknown parameter D, in case of monocular supervision this formulation jointly explain uncertainty for depth and pose, both unknown variables in π. We will show how this approach leads to sub-optimal modelling and how to overcome this limitation with the next approach.

Self-Teaching (*Self*). In order to decouple depth and pose when modelling uncertainty, we propose to source a direct form of supervision from the learned model itself. By training a first network in a self-supervised manner, we obtain a network instance $\mathcal{T}$ producing a noisy distribution $D_{\mathcal{T}}$. Then, we train a second instance of the same model, namely $\mathcal{S}$, to mimic the distribution sourced from $\mathcal{T}$. Typically, teacher-student frameworks [360] applied to monocular depth estimation [226] deploy a complex architecture to supervise a more compact one. In contrast, in our approach the teacher $\mathcal{T}$ and the student $\mathcal{S}$ share the same architecture and for this reason we refer to it as Self-Teaching (*Self*). By assuming an $L1$ loss, we can model for instance negative log-likelihood minimization as

$$\mathcal{L}_{Self} = \frac{|\mu(D_{\mathcal{S}}) - D_{\mathcal{T}}|}{\sigma(D_{\mathcal{S}})} + \log \sigma(D_{\mathcal{S}}) \tag{10.13}$$

We will show how with this strategy i) we obtain a network $\mathcal{S}$ more accurate than $\mathcal{T}$ and ii) in case of monocular supervision, we can decouple depth from pose and achieve a much more effective uncertainty estimation. Figure 10.3 summarizes our proposal.

10.1.4 Bayesian estimation

Finally, in Bayesian deep learning [135], the model uncertainty can be explained by marginalizing over all possible w rather than choosing a point estimate. According to Neal [207], an approximate solution can be obtained by sampling N models and by modelling mean and variance as

$$p(D^*|\mathcal{I}, \mathcal{D}) \approx \sum_{i=1}^{N} p(D^*|\Theta(\mathcal{I}, w_i)) \tag{10.14}$$

If mean and variance are modelled for each w_i sampling, we can obtain overall mean and variance as reported in [135, 119]

$$\mu(D) = \frac{1}{N}\sum_{i=1}^{N} \mu_i(D_i) \tag{10.15}$$

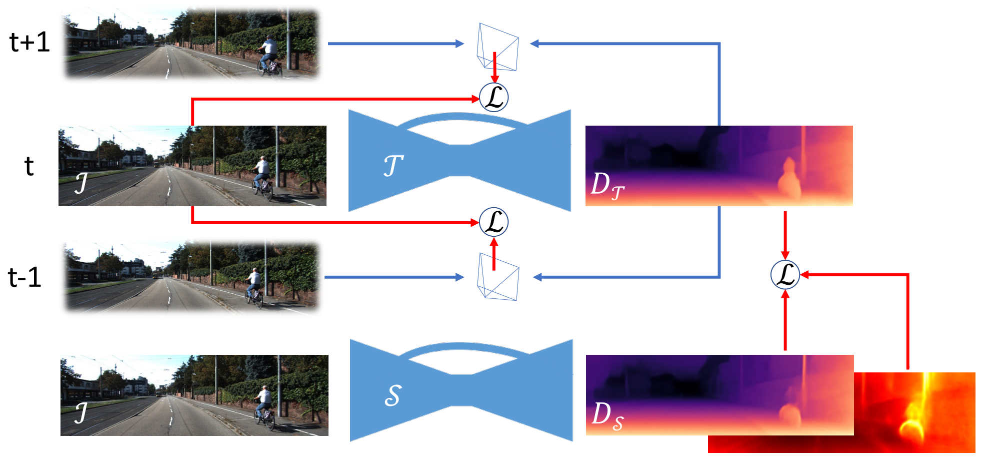


FIGURE 10.3: **Self-Teaching scheme.** A network $\mathcal{T}$ is trained in self-supervised fashion, e.g. on monocular sequences $[t-1, t, t+1]$. A new instance $\mathcal{S}$ of the same is trained on $D_{\mathcal{T}}$ output of $\mathcal{T}$.

$$\sigma^2(D) = \frac{1}{N}\sum_{i=1}^{N}(\mu_i(D_i) - \mu(D))^2 + \sigma_i^2(D_i) \tag{10.16}$$

The implementation of this approximation is straightforward by combining empirical and predictive methods [135, 119]. Purposely, in our experiments we will pick the best empirical and predictive methods, e.g. combining *Boot* and *Self* (*Boot+Self*).

10.2 Experimental results

In this section, we exhaustively evaluate self-supervised strategies for joint depth and uncertainty estimation.

10.2.1 Evaluation protocol, dataset and metrics

At first, we describe all details concerning training and evaluation.

Architecture and training schedule. We choose as baseline model Monodepth2 [80], thanks to the code made available and to its possibility to be trained seamlessly according to monocular, stereo, or both self-supervision paradigms. In our experiments, we train any variant of this method following the protocol defined in [80], on batches of 12 images resized to 192×640 for 20 epochs starting from pre-trained encoders on ImageNet [58]. Moreover, we always follow the augmentation and training practices described in [80]. Finally, to evaluate *Post* we use the same weights made publicly available by the authors. Regarding empirical methods, we set N to 8 and the number of cycles C for *Snap* to 20. We randomly extract 25% of the training set for each independent network in *Boot*. Dropout is applied after convolutions in the decoder only. About predictive models, a single output channel is added in parallel to depth prediction channel.

Depth metrics. To assess depth accuracy, we report three out of seven standard metrics defined in [65]. Specifically, we report the absolute relative error (Abs Rel), root mean square error (RMSE), and the amount of inliers ($\delta < 1.25$). They enable a compact evaluation concerning both relative (Abs Rel and $\delta < 1.25$) and absolute (RMSE) errors. Moreover, we also report the number of training iterations (#Trn), parameters (#Par), and forwards (#Fwd) required at testing time to estimate depth. In the case of monocular supervision, we scale depth as in [365].

Uncertainty metrics. To evaluate how significant the modelled uncertainties are, we compute the Area Under the Sparsification Error (AUSE, the **lower** the better) and the Area Under the Random Gain (AURG, the **higher** the better). We described these metrics in Chapter 3.2. The former quantifies how close the estimate is to the oracle uncertainty, the latter how better (or worse, as we will see in some cases) it is compared to no modelling at all. We assume Abs Rel, RMSE or $\delta \geq 1.25$ (since $\delta < 1.25$ defines an accuracy score) as ϵ.

10.2.2 Monocular (M) supervision

Depth. Table 10.1 reports depth accuracy for Monodepth2 variants implementing the different uncertainty estimation strategies when trained with monocular supervision. We can notice how, in general, empirical methods fail at improving depth prediction on most metrics, with *Drop* having a large gap from the baseline. On the other hand, *Boot* and *Snap* slightly reduce RMSE. This behaviour was consistent among multiple executions of these tests. Predictive methods as well produce worse depth estimates, except the proposed *Self* method, which improves all the metrics compared to the baseline, even when post-processed. Regarding the Bayesian solutions, both *Boot* and *Snap* performs worse when combined with *Log*, while they are always improved by the proposed *Self* method.

Uncertainty. Table 10.2 resumes performance of modelled uncertainties at reducing errors on the estimated depth maps. Surprisingly, empirical methods rarely perform better than the *Post* solution. In particular, empirical methods alone fail at performing better than a random chance, except for *Drop* that, on the other hand, produces much worse depth maps. Predictive methods perform better, with *Log* and *Self* yielding the best results. Among them, our method outperforms *Log* by a notable margin. Combining empirical and predictive methods is beneficial, often improving over single choices. In particular, *Boot+Self* achieves the best overall results.

Summary. In general *Self*, combined with empirical methods, performs better for both depth accuracy and uncertainty modelling when dealing with M supervision, thanks to disentanglement between depth and pose. We believe that empirical methods performance can be ascribed to depth scale, being unknown during training.

Method	#Trn	#Par	#Fwd	Abs Rel	RMSE	δ <1.25
Monodepth2 [80]	1×	1×	1×	0.090	3.942	0.914
Monodepth2-*Post* [80]	1×	1×	2×	0.088	3.841	0.917
Monodepth2-*Drop*	1×	1×	N×	0.101	4.146	0.892
Monodepth2-*Boot*	N×	N×	1×	0.092	3.821	0.911
Monodepth2-*Snap*	1×	N×	1×	0.091	3.921	0.912
Monodepth2-*Repr*	1×	1×	1×	0.092	3.936	0.912
Monodepth2-*Log*	1×	1×	1×	0.091	4.052	0.910
Monodepth2-*Self*	(1+1)×	1×	1×	**0.087**	3.826	**0.920**
Monodepth2-*Boot+Log*	N×	N×	1×	0.092	3.850	0.910
Monodepth2-*Boot+Self*	(1+N)×	N×	1×	0.088	**3.799**	0.918
Monodepth2-*Snap+Log*	1×	1×	1×	0.092	3.961	0.911
Monodepth2-*Snap+Self*	(1+1)×	1×	1×	0.088	3.832	0.919

TABLE 10.1: **Quantitative results for monocular (M) supervision: depth evaluation.** Evaluation on Eigen split [65] with improved ground-truth [308].

	Abs Rel		RMSE		$\delta \geq 1.25$	
Method	AUSE	AURG	AUSE	AURG	AUSE	AURG
Monodepth2-*Post*	0.044	0.012	2.864	0.412	0.056	0.022
Monodepth2-*Drop*	0.065	0.000	2.568	0.944	0.097	0.002
Monodepth2-*Boot*	0.058	0.001	3.982	-0.743	0.084	-0.001
Monodepth2-*Snap*	0.059	-0.001	3.979	-0.639	0.083	-0.002
Monodepth2-*Repr*	0.051	0.008	2.972	0.381	0.069	0.013
Monodepth2-*Log*	0.039	0.020	2.562	0.916	0.044	0.038
Monodepth2-*Self*	0.030	0.026	2.009	1.266	0.030	0.045
Monodepth2-*Boot+Log*	0.038	0.021	2.449	0.820	0.046	0.037
Monodepth2-*Boot+Self*	**0.029**	**0.028**	**1.924**	**1.316**	**0.028**	**0.049**
Monodepth2-*Snap+Log*	0.038	0.022	2.385	1.001	0.043	0.039
Monodepth2-*Snap+Self*	0.031	0.026	2.043	1.230	0.030	0.045

TABLE 10.2: **Quantitative results for monocular (M) supervision: uncertainty evaluation.** Evaluation on Eigen split [65] with improved ground-truth [308].

10.2.3 Stereo (S) supervision

Depth. In Table 10.3 we show the results of the same approaches when trained with stereo supervision. Again, *Drop* fails to improve depth accuracy, together with *Repr* among predictive methods. *Boot* produces the best improvement, in particular in terms of RMSE. Traditional *Log* improves this time over the baseline, according to RMSE and $\delta < 1.25$ metrics while, *Self* consistently improves the baseline on all metrics, although it does not outperform *Post*, which requires two forward passes.

Uncertainty. Table 10.4 summarizes the effectiveness of modelled uncertainties. This time, only *Drop* performs worse than *Post* achieving negative AURG, thus being detrimental at sparsification, while other empirical methods achieve much better results. In these experiments, thanks to the known pose of the stereo setup, *Log* deals only with depth uncertainty and thus performs extremely well. *Self*, although allowing for more accurate depth as reported in Table 10.3, ranks second this time. Considering Bayesian implementations, again, both *Boot* and *Snap* are always improved. Conversely, compared to the M case, *Log* this time consistently outperforms *Self* in any Bayesian formulation.

Method	#Trn	#Par	#Fwd	Abs Rel	RMSE	δ <1.25
Monodepth2 [80]	1×	1×	1×	0.085	3.942	0.912
Monodepth2-*Post* [80]	1×	1×	2×	0.084	3.777	**0.915**
Monodepth2-*Drop*	1×	1×	N×	0.129	4.908	0.819
Monodepth2-*Boot*	N×	N×	1×	0.085	**3.772**	0.914
Monodepth2-*Snap*	1×	N×	1×	0.085	3.849	0.912
Monodepth2-*Repr*	1×	1×	1×	0.085	3.873	0.913
Monodepth2-*Log*	1×	1×	1×	0.085	3.860	**0.915**
Monodepth2-*Self*	(1+1)×	1×	1×	0.084	3.835	**0.915**
Monodepth2-*Boot+Log*	N×	N×	1×	0.085	3.777	0.913
Monodepth2-*Boot+Self*	(1+N)×	N×	1×	0.085	3.793	0.914
Monodepth2-*Snap+Log*	1×	1×	1×	**0.083**	3.833	0.914
Monodepth2-*Snap+Self*	(1+1)×	1×	1×	0.086	3.859	0.912

TABLE 10.3: **Quantitative results for stereo (S) supervision: depth evaluation.** Evaluation on Eigen split [65] with improved ground-truth [308].

	Abs Rel		RMSE		$\delta \geq 1.25$	
Method	AUSE	AURG	AUSE	AURG	AUSE	AURG
Monodepth2-*Post*	0.036	0.020	2.523	0.736	0.044	0.034
Monodepth2-*Drop*	0.103	-0.029	6.163	-2.169	0.231	-0.080
Monodepth2-*Boot*	0.028	0.029	2.291	0.964	0.031	0.048
Monodepth2-*Snap*	0.028	0.029	2.252	1.077	0.030	0.051
Monodepth2-*Repr*	0.040	0.017	2.275	1.074	0.050	0.030
Monodepth2-*Log*	0.022	0.036	0.938	2.402	**0.018**	0.061
Monodepth2-*Self*	0.022	0.035	1.679	1.642	0.022	0.056
Monodepth2-*Boot+Log*	**0.020**	**0.038**	**0.807**	**2.455**	**0.018**	**0.063**
Monodepth2-*Boot+Self*	0.023	0.035	1.646	1.628	0.021	0.058
Monodepth2-*Snap+Log*	0.021	0.037	0.891	2.426	**0.018**	0.061
Monodepth2-*Snap+Self*	0.023	0.035	1.710	1.623	0.023	0.058

TABLE 10.4: **Quantitative results for stereo (S) supervision: uncertainty evaluation.** Evaluation on Eigen split [65] with improved ground-truth [308].

10.2.4 Monocular+Stereo (MS) supervision

Depth. Table 10.5 reports the behavior of depth accuracy when monocular and stereo supervisions are combined. In this case, only *Self* consistently outperforms the baseline and is competitive with *Post*, which still requires two forward passes. Among empirical methods, *Boot* is the most effective. Regarding Bayesian solutions, those using *Self* are, in general, more accurate on most metrics, yet surprisingly worse than *Self* alone.

Uncertainty. Table 10.6 shows the performance of the considered uncertainties. The behavior of all variants is similar to the one observed with stereo supervision, except for *Log* and *Self*. We can notice that *Self* outperforms *Log*, similarly to what observed with M supervision. It confirms that pose estimation drives *Log* to worse uncertainty estimation, while *Self* models are much better thanks to the training on proxy labels produced by the Teacher network. Concerning Bayesian solutions, in general, *Boot* and *Snap* are improved when combined with both *Log* and *Self*, with *Self* combinations typically better than their *Log* counterparts and equivalent to standalone *Self*.

Summary. The evaluation with monocular and stereo supervision confirms that when the pose is estimated alongside with depth, *Self* proves to be a better solution compared to *Log* and, in general, other approaches to model uncertainty. Finally, empirical methods alone behave as for experiments with stereo supervision, confirming that the knowledge of the scale during training is crucial to the proper behavior of *Drop*, *Boot* and *Snap*.

Method	#Trn	#Par	#Fwd	Abs Rel	RMSE	δ <1.25
Monodepth2 [80]	1×	1×	1×	0.084	3.739	0.918
Monodepth2-*Post* [80]	1×	1×	2×	**0.082**	**3.666**	**0.919**
Monodepth2-*Drop*	1×	1×	N×	0.172	5.885	0.679
Monodepth2-*Boot*	N×	N×	1×	0.086	3.787	0.910
Monodepth2-*Snap*	1×	N×	1×	0.085	3.806	0.914
Monodepth2-*Repr*	1×	1×	1×	0.084	3.828	0.913
Monodepth2-*Log*	1×	1×	1×	0.083	3.790	0.916
Monodepth2-*Self*	(1+1)×	1×	1×	0.083	3.682	**0.919**
Monodepth2-*Boot+Log*	N×	N×	1×	0.086	3.771	0.911
Monodepth2-*Boot+Self*	(1+N)×	N×	1×	0.085	3.704	0.915
Monodepth2-*Snap+Log*	1×	1×	1×	0.084	3.828	0.914
Monodepth2-*Snap+Self*	(1+1)×	1×	1×	0.085	3.715	0.916

TABLE 10.5: **Quantitative results for monocular+stereo (MS) supervision: depth evaluation.** Evaluation on Eigen split [65] with improved ground-truth [308].

	Abs Rel		RMSE		$\delta \geq 1.25$	
Method	AUSE	AURG	AUSE	AURG	AUSE	AURG
Monodepth2-*Post*	0.036	0.018	2.498	0.655	0.044	0.031
Monodepth2-*Drop*	0.103	-0.027	7.114	-2.580	0.303	-0.081
Monodepth2-*Boot*	0.028	0.030	2.269	0.985	0.034	0.049
Monodepth2-*Snap*	0.029	0.028	2.245	1.029	0.033	0.047
Monodepth2-*Repr*	0.046	0.010	2.662	0.635	0.062	0.018
Monodepth2-*Log*	0.028	0.029	1.714	**1.562**	0.028	0.050
Monodepth2-*Self*	**0.022**	0.033	**1.654**	1.515	**0.023**	0.052
Monodepth2-*Boot+Log*	0.030	0.028	1.962	1.282	0.032	0.051
Monodepth2-*Boot+Self*	0.023	0.033	1.688	1.494	**0.023**	**0.056**
Monodepth2-*Snap+Log*	0.030	0.027	2.032	1.272	0.032	0.048
Monodepth2-*Snap+Self*	0.023	**0.034**	1.684	1.510	**0.023**	0.055

TABLE 10.6: **Quantitative results for monocular+stereo (MS) supervision: uncertainty evaluation.** Evaluation on Eigen split [65] with improved ground-truth [308].

10.2.5 Sparsification curves

In order to further outline our findings, we report in Figure 10.4 RMSE and Abs Rel sparsification error curves, averaged over the test set, when training with M, S or MS supervision. The plots show that methods leveraging on *Self* (blue) are the best to model uncertainty when dealing with pose estimation, i.e. M and MS, while those using *Log* (green) are better when training on S.

10.2.6 Qualitative results

We report in Figure 10.5 some qualitative examples of both depth and uncertainty maps obtained by the evaluated methods using images from KITTI. Depth maps are encoded with MAGMA color map, i.e. the warmer the color the closer the point, while uncertainty is encoded with HOT color scheme, thus the more yellow the color the more unreliable the value.

10.3 Conclusion

In this chapter, we have thoroughly investigated uncertainty modelling in self-supervised monocular depth estimation. We have reviewed and evaluated existing techniques, as well as introduced a novel Self-Teaching (*Self*) paradigm. We have considered up to 11 strategies to estimate the uncertainty on predictions of a depth-from-mono network trained in a self-supervised manner. Our experiments highlight how different supervision strategies lead to different winners among the considered methods. In particular, among empirical methods, only Dropout sampling performs well when the scale is unknown during training (M), while it is the only one failing when scale is known (S, MS). Empirical methods are affected by pose estimation, for which log-likelihood maximization gives sub-optimal results when the pose is unknown (M, MS). In these latter cases, potentially the most appealing for practical applications, the proposed *Self* technique results in the best strategy to model uncertainty. Moreover, uncertainty estimation also improves depth accuracy consistently, with any training paradigm.

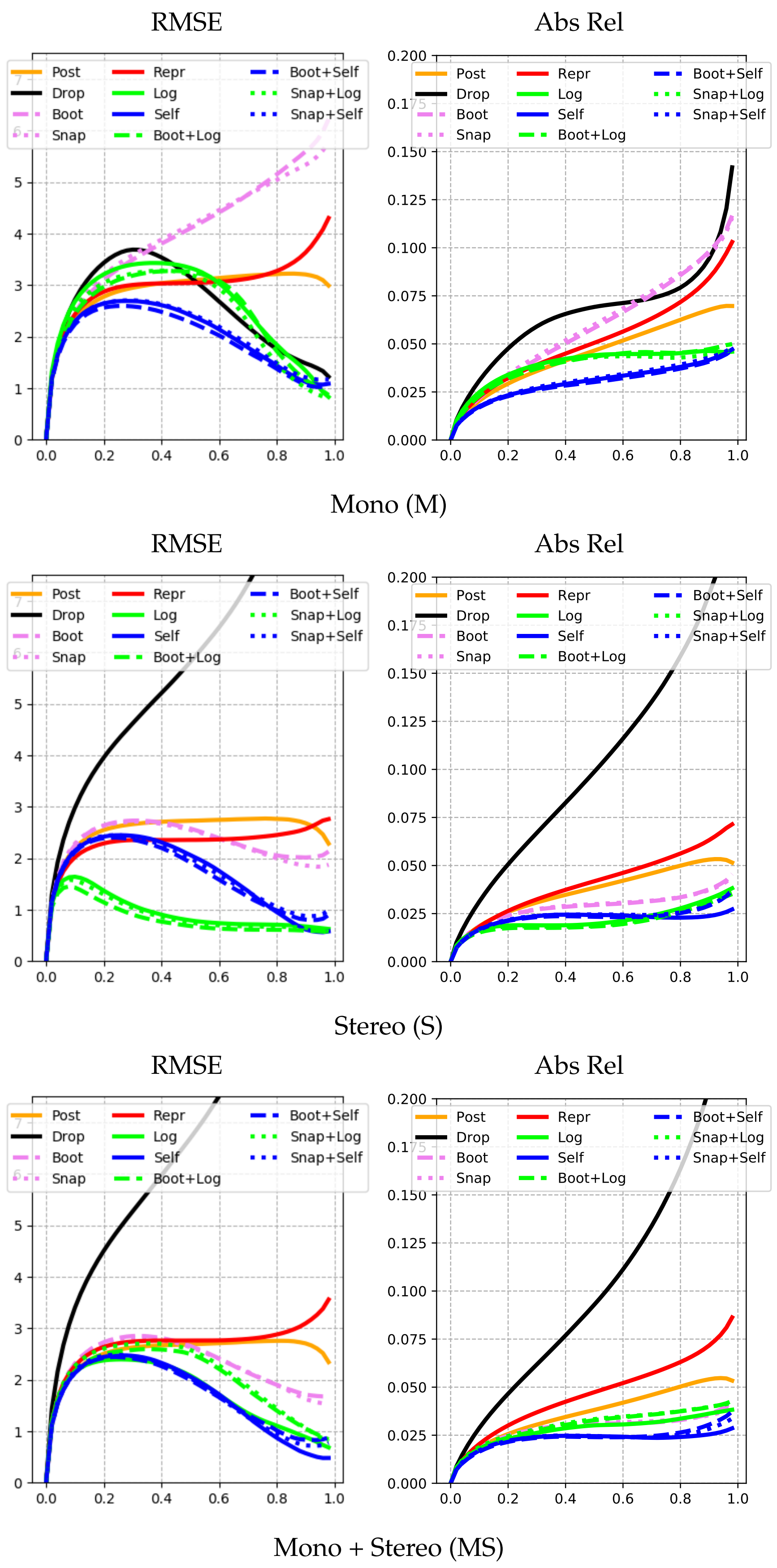


FIGURE 10.4: **Sparsification Error curves.** We report sparsification curves for Mono, Stereo and Mono+Stereo configurations. From left to right, sparsification curves for RMSE and Abs Rel error metrics. Best viewed with colors.

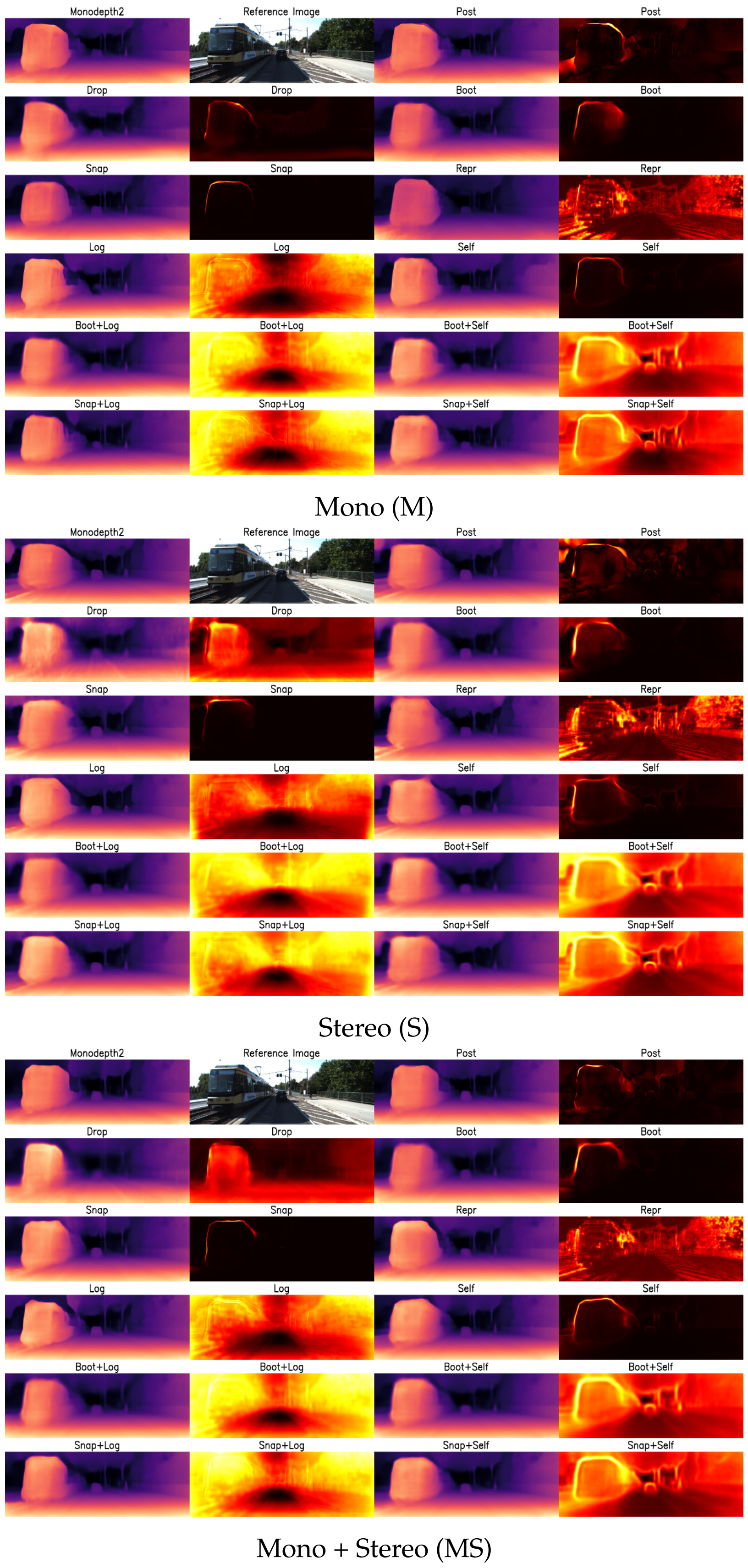


FIGURE 10.5: **Qualitative results on KITTI.** We show the depth and the relative uncertainty maps for different configurations.

Chapter 11

Improving off-the shelf disparity maps with deep learning

Chapter 9 illustrated a strategy aimed at training robust stereo models starting from raw stereo pairs. The problem of generalization is important for any method, but for strategy based on learning, it becomes crucial. In fact, models can learn priors and behaviours driven by the training data, which can be biased and non representative for the data we might face at test time. When this problem occurs, we can train the model again on a new dataset, or we can adopt robust training procedures or components devoted to tackle the issue. For instance, Zhang et al. [355] propose a normalization layer that helps stereo models in being invariant. This module represents an effective attempt towards robust out-domain deep models, but there is still much work to do. Conversely, classical methods are in general less accurate than deep learning strategies when tested in-domain, yet prove to more robust out-domain. However, classical methods are usually filtered and contain holes in predictions, while we might be interested in dense maps. Can we take obtain the best from both the worlds? In other terms: can we achieve out-domain robustness of classical methods, while exploiting the power of deep learning to achieve dense yet accurate maps?

In this chapter we introduce a novel architecture for neural disparity refinement aimed at improving disparity maps from black-box stereo methods. Specifically, to address the generalization issue of learning-based stereo strategies, we propose to deploy a traditional, domain-agnostic stereo algorithm in order to provide the input disparity map to our neural refinement module. Thereby, we realize an hybrid handcrafted-learned solution that neatly outperforms state-of-the-art deep networks when processing previously unseen image content. Moreover, the refined disparity maps predicted by our framework are sharp at depth discontinuities, which is another popular issue shared by other methods. We achieve this result thanks to a novel loss function that allows for expressing the output disparities as a combination of a categorical value and a continuous offset. As a result, point clouds from our depth maps are less affected by *flying pixels* and more realistic. Finally, the peculiar continuous formulation allows to estimate a refined map at any arbitrary output resolution. This is particularly important in the field of mobile devices: in fact, many smartphones are equipped with multiple cameras with different resolutions, which is different than the conventional stereo setup. Our framework, instead, elegantly handles the problem, allowing to work with both *balanced* and *unbalanced* configurations. Figure 11.1 depicts an example of unbalanced stereo, showing the benefits of our framework.

This chapter is based on the paper Neural Disparity Refinement for Arbitrary Resolution Stereo [10].

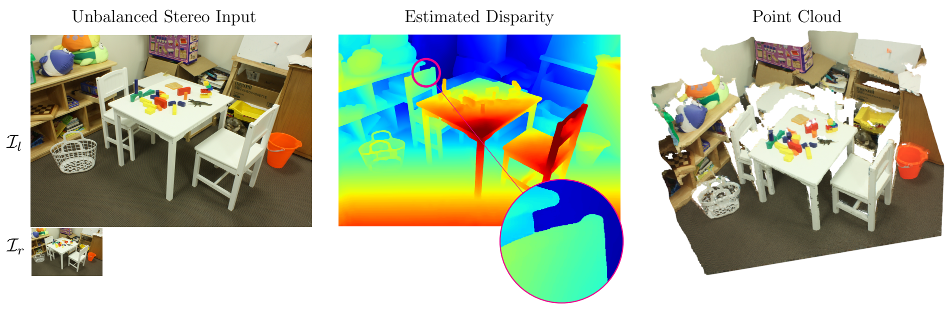


FIGURE 11.1: **Example of Arbitrary Resolution Stereo.** Given an **unbalanced** stereo pair made of a high-res image, $\mathcal{I}_l$, with shape 2724×1848 and a low-res image, $\mathcal{I}_r$, at 691×462, both unseen and real, our method can estimate a high-res disparity map at 2742×1848 with sharp edges (leading to a clean point cloud near depth discontinuities) based on a single **training on synthetic data only**.

11.1 Proposed Architecture

In this section, we introduce our Neural Disparity Refinement architecture that, given an input disparity map computed by any black-box stereo method, allows for refining it at any desired output resolution, e.g. *higher* than the input one. To this aim, we propose a simple yet effective neural architecture that accepts as inputs the reference RGB image of a stereo pair alongside a corresponding noisy disparity map, the latter possibly even at a lower resolution than the former. A standard convolutional neural network extracts and combines deep features computed from both inputs. Then, the final features are feed into two Multi-Layer Perceptrons (MLPs) that, thanks to a continuous formulation, allows for estimating a refined disparity map at any chosen resolution. The whole network is trained end-to-end based on a novel loss function that enables to predict sharp and precise disparities at object boundaries. In the remainder of this section we explain in detail the key components of our proposed architecture.

11.1.1 Continuous Disparity Refinement Network

Our network implements three different steps: i) feature extraction, ii) feature interpolation and iii) disparity prediction with subpixel refinement, as illustrated in Figure 11.2.

Feature Extraction. Given two rectified stereo images, $\mathcal{I}_l$ and $\mathcal{I}_r$, with shapes $w_l \times h_l$ and $w_r \times h_r$ and the same aspect ratio ($\frac{w_l}{h_l} = \frac{w_r}{h_r}$), we aim at obtaining a refined disparity map, $\widetilde{D}$, at any arbitrary spatial resolution $w_o \times h_o$. Depending on factor $\kappa = \frac{w_l}{w_r}$, we may have a *balanced* ($\kappa = 1$) or *unbalanced* ($\kappa \neq 1$) stereo setup. In the latter one, the rectification constraint shall be understood to hold up to a scale factor, i.e. $\mathcal{I}_l$ and $\mathcal{I}_r$ turn out to be rectified whenever resized to the same -arbitrary-shape. Hereinafter, we will also refer to κ as to *unbalance* factor.

The disparity map to be refined, D, may come from any stereo approach, either traditional or learned. In both the balanced and unbalanced setups we assume D to have the same resolution as the reference stereo image $\mathcal{I}_l$. In particular, as depicted in Figure 11.2, in case of unbalanced setup, we assume that a low-resolution disparity map $D_\downarrow$ is computed by the stereo blackbox by downsampling $\mathcal{I}_l$ to the same resolution as $\mathcal{I}_r$ and later upsampled to match the shape of $\mathcal{I}_l$. Then, two separate convolutional encoders, $\phi_\mathcal{I}$ and ϕ_D, extract features at different resolutions, collectively denoted here as $\boldsymbol{\mathcal{F}}_\mathcal{I}$ and $\boldsymbol{\mathcal{F}}_D$, from $\mathcal{I}_l$ and D, respectively. A decoding stage,

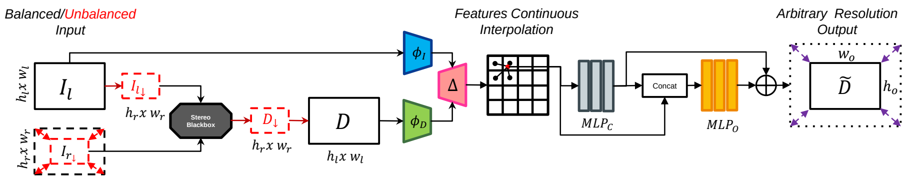


FIGURE 11.2: **Neural Disparity Refinement, architecture overview.** Given a rectified stereo pair captured using either a balanced or unbalanced (red dotted lines) stereo setting, our goal is to estimate a refined disparity map $\widetilde{D}$ at any arbitrary spatial resolution starting from noisy disparities D pre-computed by any existing stereo blackbox. We first extract deep high-dimensional features from $\mathcal{I}_l$ and D using two separate convolutional branches, $\phi_{\mathcal{I}}$ and ϕ_D, that are combined together by a decoder, Δ. Then, at each continuous 2D location in the $\mathcal{I}_l$ image domain, we interpolate features across the levels of Δ in order to feed them into a disparity estimation module realized through two MLPs, namely MLP$_C$ and MLP$_O$, which predict an integer disparity value and a sub-pixel offset, respectively.

referred to as Δ in Figure 11.2, is in charge of merging the features from the two encoders while restoring the original $\mathcal{I}_l$ resolution. At each level l of the decoder Δ, the features from the corresponding encoder levels, $\mathcal{F}^l_{\mathcal{I}}$ and $\mathcal{F}^l_D$, are aggregated (by mean of channel-wise sum) and used as skip-connections for the upsampled features from the previous decoder level, denoted here as $\mathcal{F}^{l-1}_{\Delta}$.

Feature interpolation. In order to infer the refined disparity map $\widetilde{D}$ at any arbitrary resolution, similarly to [303], we formulate the disparity prediction problem as the estimation of a function defined on a continuous 2D domain. In particular, rather than directly predicting a disparity map from Δ, first we compute features across the decoder levels, $\mathcal{F}^l_{\Delta}$, at any arbitrary continuous location of the 2D image domain $\mathcal{I}_l$ by bilinearly interpolating between the four nearest discrete locations. Then, the interpolated features computed at each level are concatenated and forwarded to a Multi-Layer Perceptron (MLP) that provides the disparity prediction, $\widetilde{D}$, at the considered continuous 2D location. As the MLP predicts a disparity value based on the corresponding point-wise features, the proposed formulation allows for choosing any desired output resolution by simply sampling features at continuous spatial locations of the 2D domain.

Disparity prediction with subpixel precision. Similar to standard regression tasks, existing disparity refinement approaches adopt a $L1$ loss [78, 22, 69]. However, this choice can cause severe over-smoothing effects at depth discontinuities [46, 303], which result in bleeding artifacts when pixels are converted into a 3D point cloud. This problem, which may impact quite negatively on the downstream 3D application, is typically caused by the multi-modal disparity distributions occurring at object boundaries and the smooth function approximation yielded by standard neural network. Very recently, the over-smoothing effect has been tackled in the stereo literature by forcing the multi-modal distribution to be uni-modal [46] or by predicting bimodal mixture densities [303] aimed at modelling both the foreground and background disparities near edges. As an alternative, we leverage a simple yet effective strategy that allows us to alleviate the over-smoothing effect as well as to achieve accurate disparity estimations. In particular, as shown in Figure 11.2, for the given continuous location in the 2D domain, we deploy i) a first MLP, denoted as MLP$_C$, to predict a categorical disparity distribution by casting disparity estimation as a classification task and ii) a second MLP, namely MLP$_O$, to regress a sub-pixel offset which is added to the most likely integer disparity predicted by MLP$_C$. As depicted in Figure 11.2, both the MLPs process the features computed at the given spatial location, referred to in the following as $\mathcal{F}_\Delta$, these being further concatenated with the predicted integer disparity in order to provide the whole input to the second one. Thus, Equation 11.1 describes how the disparity $\widetilde{D}$ is predicted by our model at any given 2D location in the image:

$$\widetilde{D} = \theta(\text{MLP}_C(\mathcal{F}_\Delta)) + \text{MLP}_O(\mathcal{F}_\Delta) \tag{11.1}$$

where MLP_C is trained to predict a disparity distribution by a cross-entropy loss, while MLP_O predicts an offset in the range $[-1, 1]$ and is trained using an $L1$ loss. The function θ represents the argmax over the distribution predicted by MLP_C. Accordingly, the final activations of MLP_C and MLP_O are the Softmax and Tanh functions, respectively. Denoting D^* as the ground-truth disparity, the final loss function, $\mathcal{L}$, is computed as the sum of two terms:

$$\begin{aligned} \mathcal{L} = &-\mathcal{N}(D_i^*, \sigma) * \log\Big(\text{MLP}_C(\mathcal{F}_\Delta)\Big) \\ &+ \Big|\text{MLP}_O(\mathcal{F}_\Delta) - D_s^*\Big| \end{aligned} \tag{11.2}$$

where D_i^* and D_s^* denote the integer disparity closest to D^* and the difference between the integer disparity predicted by MLP_C and D^*, respectively:

$$D_i^* = \lfloor D^* + 0.5 \rfloor \tag{11.3}$$

$$D_s^* = \Big|D^* - \theta\Big(\text{MLP}_C(\mathcal{F}_\Delta)\Big)\Big| \tag{11.4}$$

while $\mathcal{N}(D_i^*, \sigma)$ is a Gaussian distribution centered at D_i^* ($\sigma = \sqrt{2}$ in our experiments).

11.2 Experiments

In this section, we first describe the datasets adopted to validate our proposal. Then, we carry out extensive experiments to demonstrate the benefits brought in by the proposed neural disparity refinement approach when addressing both balanced as well as unbalanced stereo settings.

Appendix D provides further training details and additional qualitative examples, and it also describes the protocol required for the calibration and rectification stages of real unbalanced stereo rigs.

11.2.1 Datasets

For our experiments, we rely on SceneFlow, KITTI, Middlebury v3, ETH3D and UnrealStereo4K datasets. Since Chapter 3.1 provides a detailed description for all of them, we now recap briefly only important details.

SceneFlow. The SceneFlow dataset [191] is a synthetic dataset containing around 35k low-resolution (960 × 540) stereo pairs. In our experiments, we use 22340 stereo pairs for training, 50 for validation and 387 for testing.

KITTI. The KITTI dataset [196] is a low-res (~ 0.4 Mpx) real-world stereo dataset. We rely on KITTI 2012 [77] (194 training and 195 testing stereo pairs) and KITTI 2015 [196] (200 training and 200 testing pairs) versions.

Middlebury v3. The Middlebury v3 [263] contains high-resolution (~ 5 Mpx) real-world stereo pairs at full-res (F) or downsampled to half (H) and quarter resolution (Q).

ETH3D. The ETH3D dataset [269] counts 27 grayscale low-res (~ 0.4 Mpx) stereo pairs.

UnrealStereo4K. The UnrealStereo4K is a synthetic high-resolution (3840×2160) stereo dataset. Following [303], we use 7720 images for training, 80 for validation and 200 for testing. Given its high resolution, we employ this dataset in the unbalanced setup as it allows to simulate different κ factors.

Evaluation metrics. For evaluation, we adopt some popular stereo metrics described in Chapter 3.2. Specifically, we compute the EPE as well as the BAD metrics. Furthermore, we adopt the Soft-Edge-Error (SEE) metric to evaluate the correctness of the predicted disparities at object-boundaries, to assess the capability of a method to produce sharp depth discontinuities. In our experiments, we set the size of ground-truth patches k to 5×5 when evaluating the SEE metric.

11.2.2 Ablation study

In this section, we examine the importance of the proposed loss function and demonstrate the robustness of our network to diverse sources of input disparities.

Loss function. Table 11.1 reports the performance of our architecture trained by different loss functions. Specifically, following the balanced setup protocol described in D.1, we train our architecture using a naïve disparity regression $L1$ loss, the mixture bimodal loss proposed in [303] to handle sharp disparity discontinuities and our proposed loss function (Equation 11.2). Notice that, for both the $L1$ and the bimodal mixture output representations, a single MLP is in charge of regressing either the final disparity, for the former, or the five parameters of a univariate bimodal mixture distribution for the latter. In our architecture, we achieve this by modifying the last layer of MLP_C and dismissing MLP_O. Then, we evaluate the trained models on the test set of the SceneFlow dataset using AD-Census [351], C-CNN [179] and SGM [101] as stereo black-boxes providing the map to be refined, so as assess upon the robustness of our method to diverse sources of input disparities. Firstly, we can observe how all the trained models improve the input disparity D by a large margin regardless the adopted loss function, proving the effectiveness of our architecture on the disparity refinement task with all the considered stereo black-boxes, and even when considering a method never seen at training time, such as C-CNN [179]. Among the considered losses, the standard disparity regression produces worst results. Moreover, it is worth noticing that, although the bimodal mixture formulation and the proposed loss achieve rather similar results in terms of SEE, our output representation is consistently more accurate with respect to all the other error metrics, which vouches for the overall superiority of our proposal.

End-to-end stereo blackbox. We also investigate the capability of our architecture to refine the disparity maps predicted by state-of-the-art deep stereo networks. In particular, in Table 11.2 we consider several deep architectures trained on SceneFlow as stereo black-boxes and evaluate the refined disparities yielded by our method on the 387 SceneFlow test images. Consistently to Table 11.1, our method successfully ameliorates the initial disparity maps for all the considered stereo blackboxes on both the BAD3 and the SEE metrics, thus proving that our architecture is

Input	Method	BAD2	BAD3	BAD4	BAD5	EPE	SEE
AD-Census	D	46.23	45.79	45.46	45.20	24.68	21.78
	$L1$	13.94	9.81	7.57	6.16	1.86	3.14
	Bimodal	9.37	6.54	5.11	4.25	1.66	**1.47**
	Ours	**8.49**	**6.10**	**4.86**	**4.10**	**1.53**	1.48
C-CNN	D	27.09	24.86	23.67	22.86	13.46	10.84
	$L1$	10.54	7.69	6.14	5.14	1.68	2.86
	Bimodal	7.93	5.66	4.56	3.90	1.64	1.38
	Ours	**7.11**	**5.25**	**4.28**	**3.68**	**1.42**	**1.36**
SGM	D	23.34	21.49	20.52	19.88	9.51	8.51
	$L1$	8.86	6.39	5.13	4.32	1.37	5.63
	Bimodal	6.08	4.49	3.64	3.11	1.25	**1.17**
	Ours	**5.80**	**4.33**	**3.56**	**3.07**	**1.19**	1.26

TABLE 11.1: **Comparison between losses** on the SceneFlow test set. The task concerns refining the initial disparity map provided by different blackboxes, i.e. both handcrafted (AD-Census[351], SGM[101]) and learned (C-CNN[179]) stereo matchers. We report the results obtained by our network when trained by a standard $L1$, a bimodal mixture representation [303] and our proposed loss.

also be beneficial when deployed in conjunction with deep stereo models. Again, it is worth highlighting that none of the networks considered in Table 11.2 was used as stereo blackbox to train our architecture. Yet, it can be observed how the EPE score slightly increase in case of the highly accurate disparity maps computed by top-performing stereo networks, such as GANet and AANet. We regard this as a trade-off associated with a loss aimed at better capturing depth edges, like ours (Equation 11.2), compared to a standard regression loss. In fact, stereo networks trained by standard regression losses produce over-smoothed disparities at object-boundaries that are not penalized by the EPE metric but lead to severe bleeding artifacts when converted to 3D point clouds. Conversely, our approach takes sharp predictions at edges which, when wrong, may cause larger errors and slightly higher EPEs. However, sharp depth discontinuities result in clear and more realistic 3D point clouds, as discussed later.

11.2.3 Balanced setup

In this section, we conduct experiments considering the standard balanced stereo setup. In particular, first we compare the proposed architecture to the main existing deep networks designed to pursue disparity refinement. Then, we assess the capability to generalize to unseen data comparatively with respect to DDR [78], i.e. the only refinement network that has been evaluated also on the Middlebury v3 dataset. Finally, we compare our proposal to several stereo methods and across different real-world scenarios. This highlights how, by deploying a traditional stereo matcher [101] as blackbox, our architecture yields superior zero-shot generalization even with respect to the recent end-to-end networks specifically designed to achieve this capability.

Input	BAD3	SEE	EPE
GANet [354]	3.55	1.83	**0.95**
GANet [354]+ Ours	**3.44**	**1.43**	0.96
AANet [334]	4.12	2.81	**1.10**
AANet [334] + Ours	**3.81**	**1.62**	1.14
HSMNet [337]	8.02	3.77	1.86
HSMNet [337] + Ours	**6.13**	**1.86**	**1.53**
PSMNet [43]	7.98	2.96	1.87
PSMNet [43] + Ours	**6.94**	**1.85**	**1.71**

TABLE 11.2: **End-to-end networks as stereo blackbox.** We validate our model using disparity maps computed by several end-to-end stereo network. For all the networks, we used the official weights released by the authors after training on SceneFlow.

Comparison to existing refinement methods

Comparison to refinement strategies. We compare our proposal to the state-of-the-art published methods on the online KITTI 2015 leaderboard. In order to be compliant with the competitors, similarly to [22, 78, 127], we started from the model pre-trained on SceneFlow and then fine-tuned it by the 200 images of the KITTI 2015 training set based on the available sparse ground-truth disparities. The first part of Table 11.3 reports the results of our submission alongside those of other competing refinement methods: our architecture achieves state-of-the-art results in all metrics (D1-all, D1-fg and D1-bg), clearly outperforming all the other refinement techniques. In the second part of Table 11.3 we also report the results achieved on KITTI 2015 by several end-to-end stereo networks: it is worth highlighting how our refinement architecture yields *in-domain* performance comparable with respect to these latter.

Method	D1-all	D1-fg	D1-bg
	Disparity Refinement		
Dil-Net [69]	3.92	7.44	3.22
DRR $\times 2$ [78] †	3.16	6.04	2.58
LRCR [127]	3.03	5.42	2.55
RecResNet [22]	3.10	6.30	2.46
Ours †	**2.35**	**3.93**	**2.03**
	End-to-End Stereo		
AANet [334]	2.55	5.39	1.99
PSMNet [43]	2.32	4.62	1.86
HITNet [291]	1.98	3.20	1.74
DSMNet [355]	**1.77**	**3.23**	**1.48**

TABLE 11.3: **Evaluation on KITTI 2015 Benchmark.** Methods indicated with † consider the disparity maps computed by C-CNN [179] as noisy input of the network. The other refinement methods adopt different noisy disparity inputs as described in the papers.

Comparison with refinement frameworks. We compare our method to other refinement models [78, 22, 69]. Table 11.4 reports the results obtained on Middlebury v3 (a) and KITTI 2015 (b), (c). In the former case, we compare with DRR [78] after training on the SceneFlow dataset. For the sake of evaluation, the two methods adopt the same noisy input disparities computed by a deep patch matching approach [179] trained on KITTI. Notice that, differently from [78], at training time our model is fed only with noisy inputs extracted by traditional stereo matchers, i.e. SGM and AD-Census. Despite this, our method notably outperforms DRR by a large margin in all metrics.

In (b) and (c) we collect results on KITTI 2015, for which we fine-tune our model on the first 160 images of the training set and evaluate on the remaining 40, the same setting followed by DRR [78], RecResNet [22] and Dil-Net [69]. Again, our model at training time is fed only with inputs obtained through SGM and AD-Census, while at testing time it refines disparity maps by C-CNN [179] (b) or OpenCV SGBM implementation (c), outperforming DRR and RecResNet in the former case, Dil-Net in the latter.

Middlebury v3

	BAD2		EPE	
Method	Non-Occ	All	Non-Occ	All
C-CNN [179]	18.24	26.71	6.06	8.71
DRR [78]	12.85	17.83	1.77	2.37
DRR ×2 [78]	11.53	16.41	1.79	2.32
Ours	**10.84**	**15.02**	**1.38**	**1.84**

(a)

KITTI 2015

	BAD3		EPE	
Method	Non-Occ	All	Non-Occ	All
C-CNN [179]	6.41	8.25	1.70	2.46
DRR ×2 [78]	2.58	3.08	0.78	0.84
RecResNet [22]	-	3.46	-	-
Ours	**2.27**	**2.60**	**0.75**	**0.79**

(b)

KITTI 2015

	BAD3
Method	All
SGBM [101]	5.15
Dil-Net (ref.) [69]	4.58
Dil-Net (fus.) [69]	3.07
Ours	**2.93**

(c)

TABLE 11.4: **Comparison with refinement frameworks**. In (a), all models are trained on SceneFlow dataset and tested on the 15 images of the Middlebury v3 training dataset at quarter resolution. In (b) and (c), models are fine-tuned on the first 160 images of KITTI 2015 training set and evaluated on the remaining 40.

Zero-shot generalization

Finally, we evaluate the generalization ability of our method using three different real-world datasets (KITTI, Middlebury v3 and ETH3D). We compare our network architecture to traditional stereo techniques as well as to recent state-of-the-art end-to-end stereo networks. For fairness, all networks are trained in a supervised setting on the SceneFlow synthetic dataset only. Table 11.5 shows how our architecture, when fed with input disparity maps computed by a traditional stereo algorithm such as SGM [101], consistently achieves the highest accuracy on all the considered datasets. It is particularly remarkable how our framework outperforms even the end-to-end stereo networks specifically designed for robust cross domain generalization [355, 38].

11.2.4 Unbalanced setup

In this section, we demonstrate how our architecture can be effectively deployed to handle unbalanced stereo images. To this aim, we experiment with stereo images acquired at different resolutions by adopting the synthetic high-resolution UnrealStereo4K dataset, that allows us to simulate different unbalance factors, κ, as well as to evaluate the accuracy of the estimated disparities using ground-truth information. Similarly to the balanced setup, we also assess out-of-domain generalization performance by testing on Middlebury v3 without any fine-tuning.

	BAD3		BAD2			BAD1
Target Domain	KITTI		Middlebury v3			ETH3D
	2012	2015	Full	Half	Quarter	
SGM [101]	14.7	14.0	27.6	23.2	17.7	13.4
PSMNet [43]	27.8	30.7	39.5	25.1	14.2	23.8
GANet [354]	10.1	11.7	32.2	20.3	11.2	14.1
HITNet [291]	6.4	6.5	-	-	-	-
MS-GCNet [38]	*5.5	6.2	-	18.5	-	8.8
DSMNet [355]	6.2	6.5	21.8	13.8	8.1	6.2
SGM [101] + Ours	**6.0/*5.0**	**5.5**	**19.2**	**12.4**	**7.9**	**4.8**

TABLE 11.5: **Generalization performance.** All methods are trained on SceneFlow and tested on the KITTI, Middlebury v3, and ETH3D datasets. Errors are the percentage of pixels with EPE greater than the specified threshold. We use the standard evaluation thresholds: 3px for KITTI, 2px for Middlebury v3, 1px for ETH3D. In KITTI the results labeled with * denote that occluded pixels have not been considered in the evaluation.

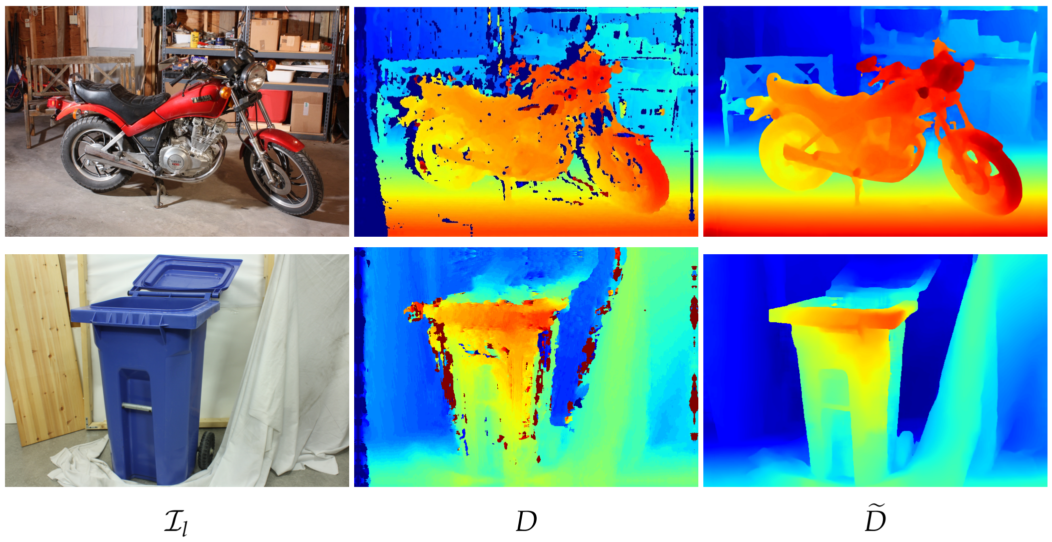


FIGURE 11.3: **Qualitative results on Middlebury v3 - balanced setup**. From left to right, the input image $\mathcal{I}_l$ from Middlebury v3, the disparity map D produced by SGM (top) and C-CNN (bottom) and our refined disparity $\widetilde{D}$.

Handling unbalanced stereo images

Table 11.6 reports our experimental study dealing with the unbalanced stereo setting on the UnrealStereo4K dataset. We assume two baseline approaches to deal with this setting, namely i) downsampling $\mathcal{I}_l$ to the same low-resolution as $\mathcal{I}_r$, estimating disparity and finally upsampling it by bilinear interpolation to the resolution of $\mathcal{I}_l$ or ii) upsampling the low-res $\mathcal{I}_r$ to the same resolution as $\mathcal{I}_l$ and directly estimating disparity at the resolution of $\mathcal{I}_l$. We run experiments starting from three stereo methods: SGM, PSMNet [43] and HSMNet [337]. According to the considered method, one approach is preferred to the other. Indeed, for SGM and PSMNet we adopt i), as both methods cannot process high-res images due to memory constraints, while for HSMNet we adopt the latter, since its architecture is specifically designed to handle high-res images. We train PSMNet and HSMNet on the official training split of the dataset, and evaluate them on the test set using $\kappa = 4, 8$ and 12. Again, we train a single instance of our neural refinement network and test its accuracy when refining the raw disparities provided by SGM, PSMNet and HSMNet.

	$\kappa = 4$		$\kappa = 8$		$\kappa = 12$	
Method	BAD3	EPE	BAD3	EPE	BAD3	EPE
	Traditional Stereo					
SGM [101]	26.33	41.56	37.74	43.27	50.69	45.45
SGM [101] + Ours	**12.21**	**5.65**	**15.92**	**7.37**	**22.06**	**8.04**
	End-to-End Stereo					
PSMNet [43]	15.22	4.37	17.83	4.67	42.69	7.42
PSMNet [43] + Ours	**12.45**	**3.86**	**14.17**	**4.06**	**35.98**	**6.22**
HSMNet [337]	15.31	6.73	29.07	9.25	43.14	13.40
HSMNet [337] + Ours	**12.12**	**5.61**	**20.36**	**6.90**	**31.67**	**9.43**

TABLE 11.6: **Experimental study of unbalanced setups.** We rely on the UnrealStereo4K dataset to evaluate different methods in unbalanced setups featuring different κ factors.

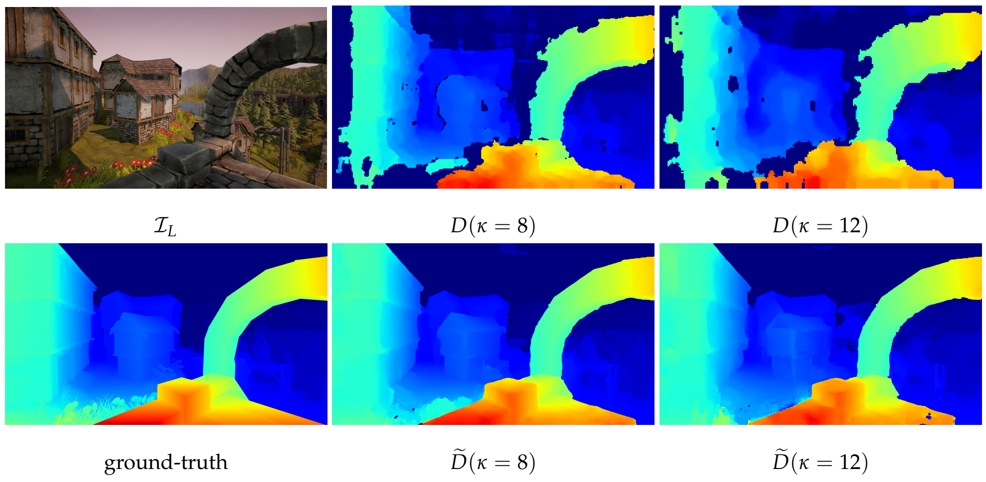


FIGURE 11.4: **Qualitative results on UnrealStereo4K – unbalanced setting.** The top row depicts the input image, $\mathcal{I}_l$, at 3840×2160 and the disparity maps, D, computed by SGM when the right image, $\mathcal{I}_r$, is 480×270 and 320×180 ($\kappa = 8$ and 12). The bottom row shows ground-truth and estimated disparity $\widetilde{D}$ at 3840×2160.

We can observe how applying our refinement strategy yields consistently a significant accuracy improvement with all the considered methods. By taking a deeper look we can also notice that applying our strategy to the three methods produces almost equivalent BAD3 results in case of the lowest unbalance factor, i.e. $\kappa = 4$. In this case, HSMNet yields the best BAD3 results. With a larger unbalance factor, as in the case of $\kappa = 12$, the number of outliers increases significantly when refining disparity maps produced by deep networks, while processing the outcome of SGM leads to the best result, with only a 10% BAD3 increase with respect to the $\kappa = 4$ setting, whereas PSMNet and HSMNet yield about +20% BAD3. Concerning EPE, refining the predictions by PSMNet consistently produces the lowest error.

Finally, we show in Figure 11.4 how our method can produce sharp and detailed high-res $\widetilde{D}$ disparity maps even in case of very unbalanced setups, such as $\kappa = 12$.

		BAD2	BAD3	SEE	EPE
$\kappa = 2$	D[101]	27.56	24.58	11.99	23.84
	US	**21.52**	**16.46**	**5.42**	**5.49**
	SF	21.91	18.01	5.63	8.13
$\kappa = 4$	D[101]	29.38	24.32	9.70	16.80
	US	25.61	**18.85**	**6.45**	6.20
	SF	**24.55**	19.14	6.77	**6.14**

TABLE 11.7: **Generalization to Middlebury v3 - unbalanced setup.** We consider $\kappa = \{2, 4\}$ and evaluate our models trained on SceneFlow (SF) and UnrealStereo4K (US) at F. Initial disparities, D, are computed by SGM at H and Q for $\kappa = 2$ and $\kappa = 4$, respectively.

Evaluation on Middlebury v3.

Eventually, we assess the performance of our method when tested on high-res images from Middlebury v3 by simulating an unbalanced configuration. Table 11.7 reports the results provided by neural refinement architecture trained either on SceneFlow (SF) or UnrealStereo4K (US), with both models run without any fine-tuning and evaluated using the available ground-truth at full resolution (F). The initial input disparities, D, are computed by SGM and downsampling $\mathcal{I}_l$ by a factor $\kappa = \{2, 4\}$ to match the resolution of $\mathcal{I}_r$. We point out that the model trained on SF is exactly the same as that used for the experiments in the balanced setting (Tabs. 11.3, 11.4 and 11.5). Thus, this model is able to consistently improve the accuracy of D in the unbalanced setting as well, proving that the proposed approach allows for effectively addressing arbitrary resolution stereo with a single neural network trained only once. Besides, training our neural network on US, which features higher-res images compared to SF, yields an increase in accuracy for $\kappa = 2$, i.e. when the resolution of $\mathcal{I}_r$ is closer to that of $\mathcal{I}_l$ and SGM runs on higher-resolution images. With $\kappa = 4$, conversely, training our network on either of the two datasets yields equivalent results, with both models capable of ameliorating significantly the input disparities computed by SGM.

11.2.5 Additional benefits of the proposed framework

We now provide some qualitative examples to better visualize other two benefits of the proposed formulation: sharp edges and continuous formulation.

To better appreciate the sharpness of the disparity maps predicted by our strategy, and to highlight the importance of this result in practice, in Figure 11.5 we show 3D point clouds obtained from raw disparity maps estimated by various methods. Then, we visualize also the point clouds produced using refined disparities, i.e. each disparity is given as input to our method before constructing the point cloud. From top to bottom, we show the point clouds obtained when using HSMNet [337], AANet [334] and GANet [354]. We can notice how the refinement removes most of the flying pixels introduced by the original, over-smoothed predictions. This is an important result for those applications that require accurate 3D reconstructions.

Instead, in Figure 11.6 we show a comparison between disparity maps yielded by our continuous formulation with respect to what was obtained through nearest-neighbour interpolation, highlighting once more the finer details produced by our model when dramatically increasing the resolution up to 80Mpx.

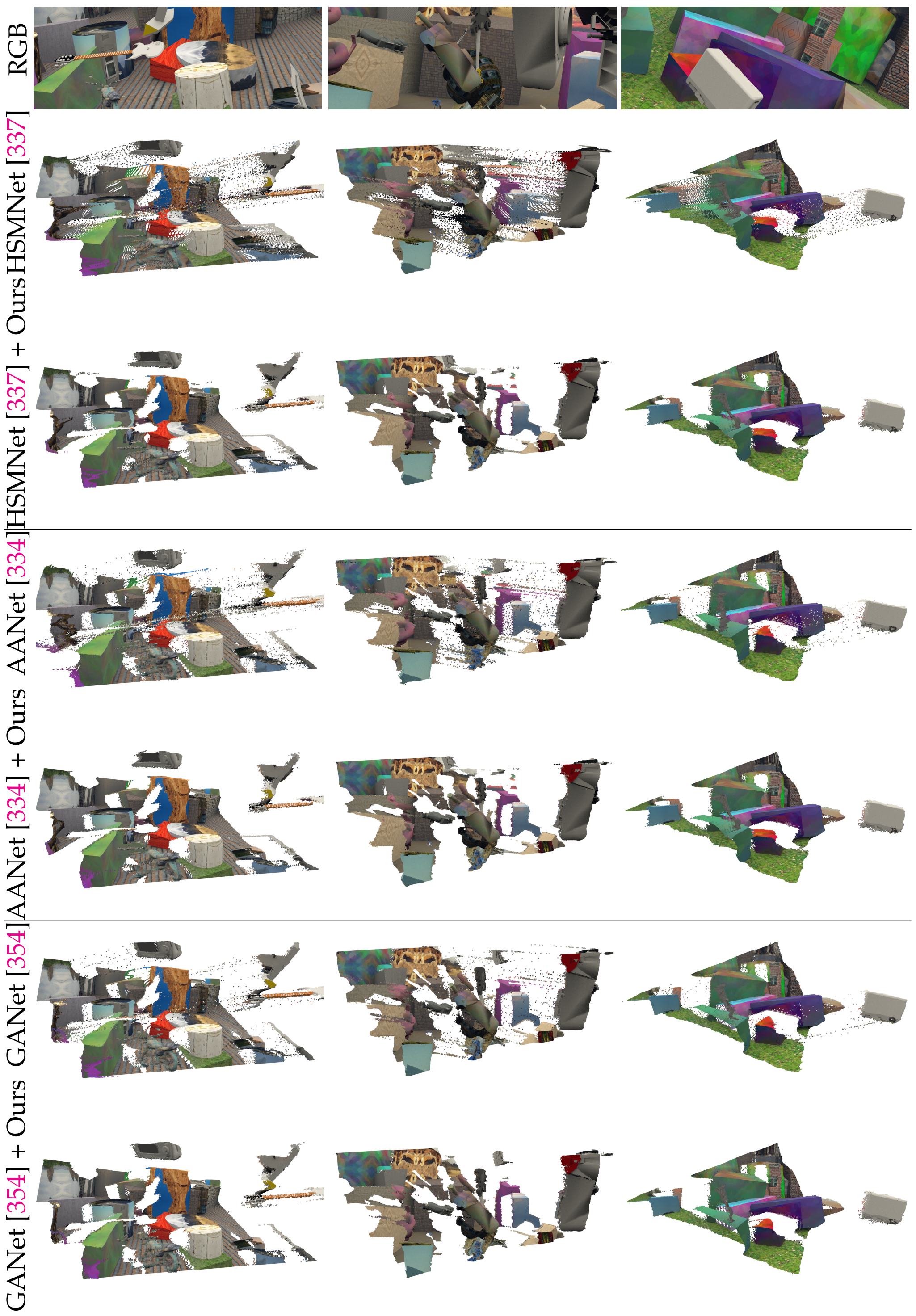


FIGURE 11.5: **Point cloud Comparison** on the SceneFlow dataset. We show the outcomes of different state-of-the-art deep stereo networks and the point clouds obtained using our refinement method on the initial disparity estimates. Note how our network allows us to notably alleviate the bleeding effect at edge boundaries, thus resulting in more accurate 3D reconstructions. Please zoom in for details.

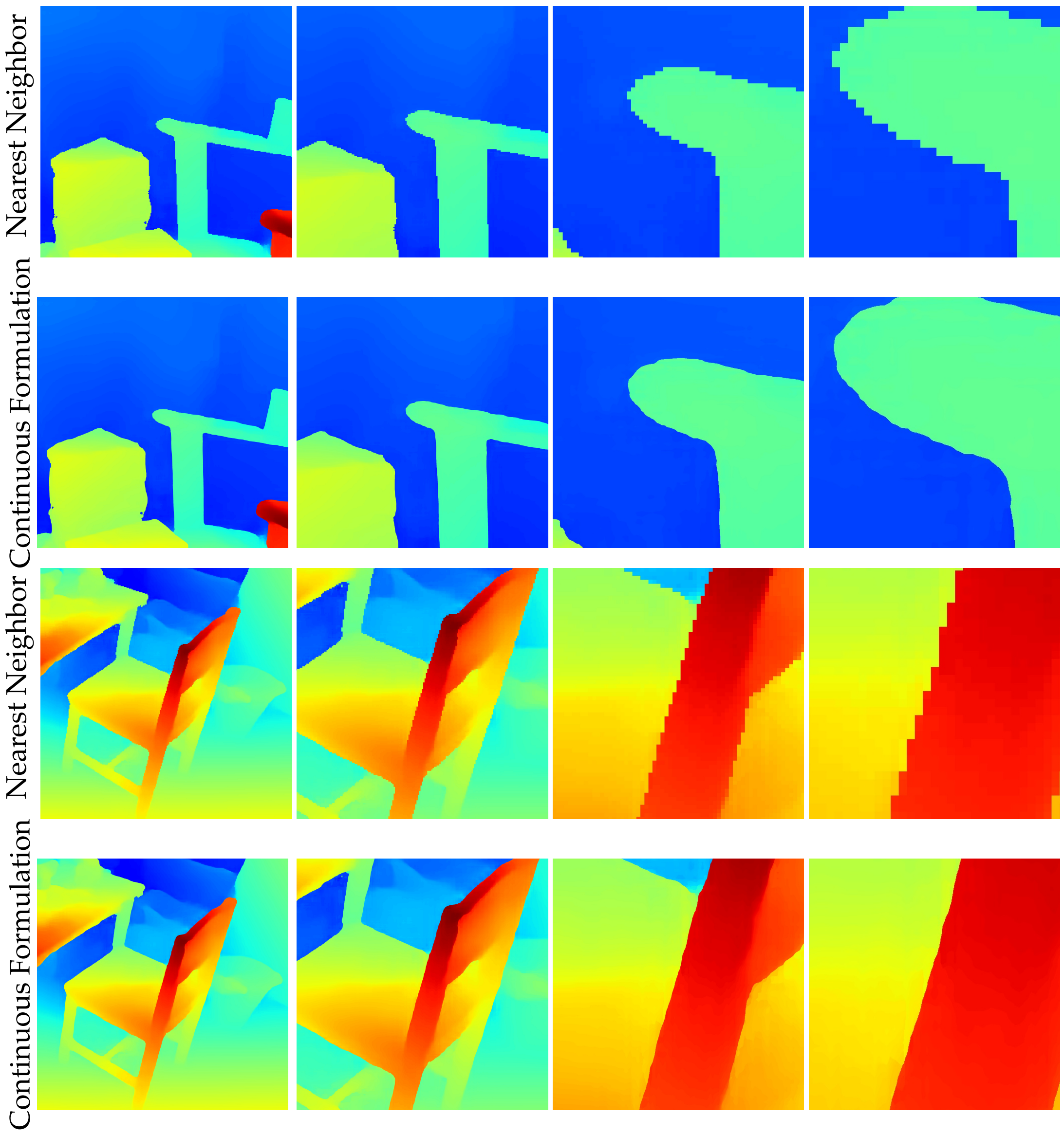


FIGURE 11.6: **Upsampling comparison**. Here, we qualitatively show the effectiveness of our continuous formulation compared to the traditional nearest-neighbor interpolation. In particular, given two noisy disparity maps at low resolution ($\sim$ 0.3 Mpx) as input of our network, we refine and upsample them by adopting an upsampling factor of 16 ($\sim$ 80 Mpx). It can be observed how our formulation allows us to obtain more precise disparities at object boundaries.

11.3 Conclusions

This chapter illustrated a novel, versatile neural architecture aimed at refining the disparity maps generated by stereo methods. Given as inputs an image and a disparity map, our network, trained once and solely on synthetic data, can refine it more accurately than other deep refinement approaches. Notably, our proposal can yield outstanding zero-shot generalization by refining disparity maps obtained by a traditional stereo matching algorithm like SGM. In particular, we have shown superior accuracy with respect to end-to-end approaches specifically conceived to excel in out-of-domain performance. Thanks to the proposed continuous formulation of the disparity refinement problem, our architecture can process effectively unbalanced stereo pairs as well as predict output disparity maps at arbitrary resolution with sharp edges.

Chapter 12

Monocular depth estimation for social distance monitoring

We now introduce a real use-case for monocular depth networks at scale, addressing a significant problem for our current society: monitoring social distancing violations. Indeed, the recent pandemic emergency raised many challenges regarding the countermeasures aimed at containing the virus spread, and constraining the minimum distance between people resulted in one of the most effective strategies. The underlining problem, consisting of monitoring people's behavior, is not new in the literature [42] and is at the core of many business intelligence tasks for data analytics [241, 94, 272, 247, 243, 14]. Different technologies [208, 209] allow to implement mechanisms to this end, often leveraging mobile or wearable devices [148, 144, 256, 128, 26] to track users' behavior. Computer vision [66, 6, 274, 5] represents a valid alternative to determine the *visual* social distance [56] in a potentially less intrusive way. They do not require explicit commitment by the people to carry mobile/wearable devices and can be adopted using standard cameras deployed for video surveillance purposes. Since extracting this kind of metric information from raw RGB images in the absence of other devices is challenging, most approaches exploit known properties of the scene and context under examination. For instance, it is often common to leverage homography-based localization [66, 5, 274] by estimating the ground plane over which people walk or by considering the overlap between detected people bounding boxes [6]. Although this allows to potentially monitor social distancing from single, uncalibrated images [5], it makes some strong assumptions on the framed scenes. For instance, constraining the task to the necessity of estimating the homography concerning the ground plane makes it impossible to monitor environments where such a plane is not visible. Such a situation frequently occurs, for instance, during many public events such as football games, theater shows or academic lectures, as shown in Figure 12.1 (b). Moreover, in order to compensate for the absence of metric knowledge, existing approaches leverage heuristics based on statistics such as average people height [5]. However, these strategies are not robust in the presence of people whose height deviates significantly from the average or when they are not orthogonally standing on the ground, e.g. a person sitting. Similarly, using the overlap between bounding boxes [6] is incline to failures due to the perspective.

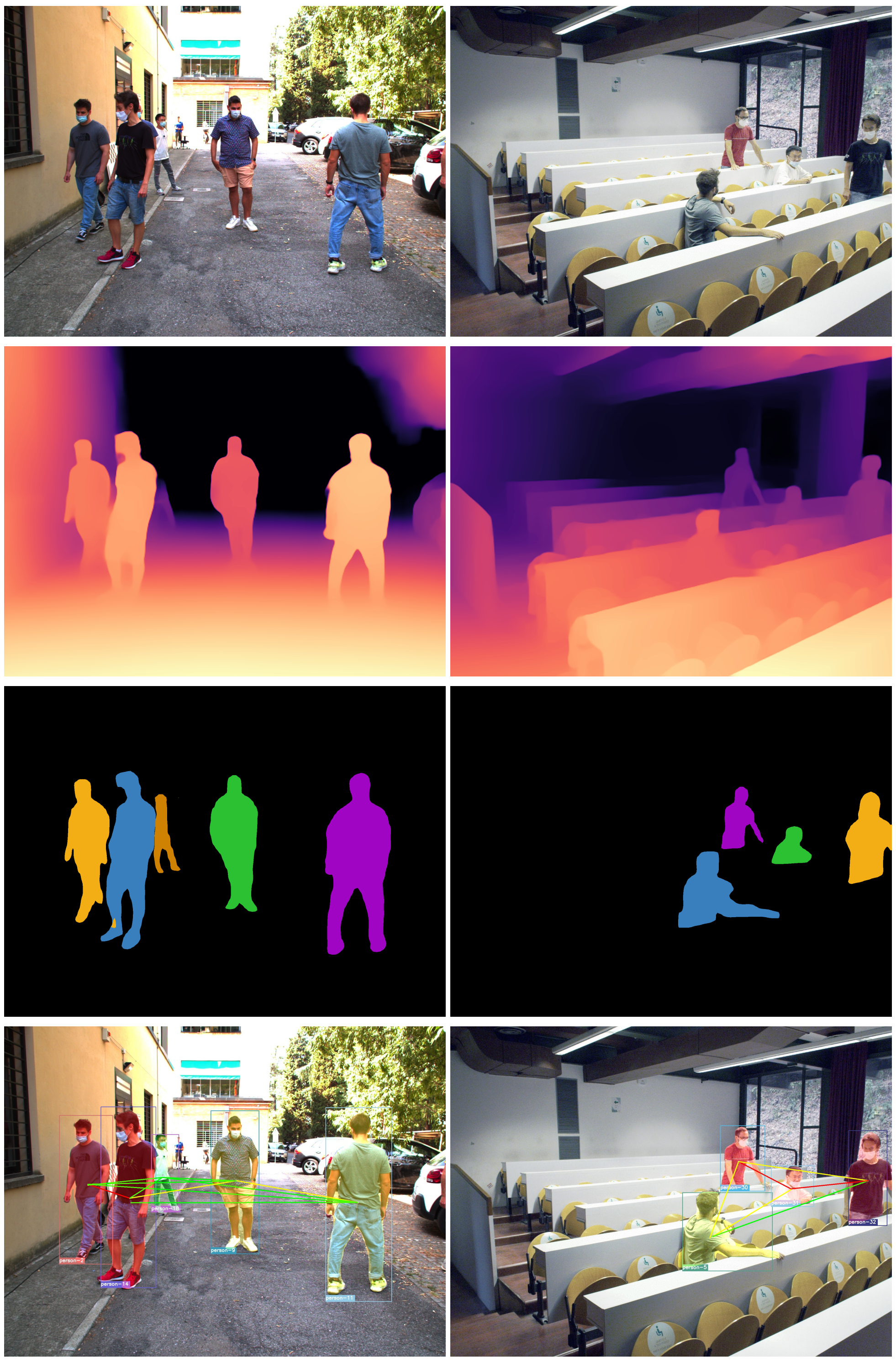


FIGURE 12.1: **Monitoring social distancing from images.** Images concerning outdoor (a) and indoor (b) environments. From top to bottom: reference RGB image, estimated depth, segmented people, and estimated inter-personal distances, with the color (red, orange, green) of the segment denoting the *class of risk* (dangerous , risky , safe), as explained in Section 12.3.4. In contrast to most other methods, our framework reliably estimates the inter-personal distance without constraints about the sensed environment even when dealing with multiple and not visible ground planes (b).

In this chapter we propose a pipeline suited for monitoring social distancing through single image depth estimation, thus requiring only a camera available in most settings. We only assume that, during system installation, an operator can use for a few seconds a standard smartphone to infer a sparse set of scaled 3D points – leveraging a scale-aware Simultaneous Localization And Mapping (SLAM) approach [81, 16] – overlapping with the area sensed by the fixed camera. Once remapped to the reference system of the fixed camera, such sparse points will be used to scale the relative monocular depth to absolute measures facing the previously mentioned issues. The procedure outlined is fast and is required once the system is initialized for the first time. Indeed, such a procedure shall be repeated only whether a large portion of the 3D structure of the background changes. After the calibration, our system leverages an off-the-shelf and pre-trained single image depth estimator [245, 244, 230] to estimate relative depth maps that we rescale according to the few known 3D points obtained during the setup stage. This strategy allows for obtaining dense, metric depth measurements for the entire scene. Finally, people are segmented through an instance segmentation network [96] and their inter-personal distance is estimated according to their 3D position in space. The content of this chapter is based on the article Monitoring social distancing with single image depth estimation [201].

12.1 Social distance monitoring

Measuring the inter-personal distance is a crucial task to monitor social distancing and thus prevent the spread of the COVID-19 pandemic. Several approaches arose in the last years [208, 209], exploiting different technologies. A popular choice consists of using dedicated devices sharing information utilizing a network [148, 144, 256, 128, 26]. However, these solutions require an ad-hoc infrastructure and a known communication protocol between the users and the backend. Unfortunately, this constraint cannot be enforced in many circumstances, such as in crowded places (e.g. stations), and require a direct collaboration of the monitored users (which have, for instance, to install an app on their mobile phone or to wear a custom Bluetooth device). Conversely, passive technologies – and specifically vision-based solutions – enable to monitor the distance among sensed people without a direct and voluntary collaboration from the users, potentially exploiting already available infrastructures such as surveillance cameras. Since the system has to work in new environments, it is common to many proposed strategies [74] a preliminary calibration phase, during which some features of the environment are detected – e.g. the principal plane in the scene, such as the floor [66, 5]. On the contrary, the proposed method aims at solving the task through direct depth estimation, requiring an initial calibration phase only to obtain control points in the static background scene – not constrained to belong to the ground plane – necessary to retrieve the metric scale of the scene. Nevertheless, our method has no additional constraints, such as, for instance, the need for a non-occluded contact point between each person to track and the plane at test time.

12.2 Proposed method

This section describes our framework, providing a detailed description of each component for social distance monitoring with a single static camera. Before its deployment, it requires a quick offline calibration that we carry out with a standard smartphone equipped with a single camera in our experiments. Once completed,

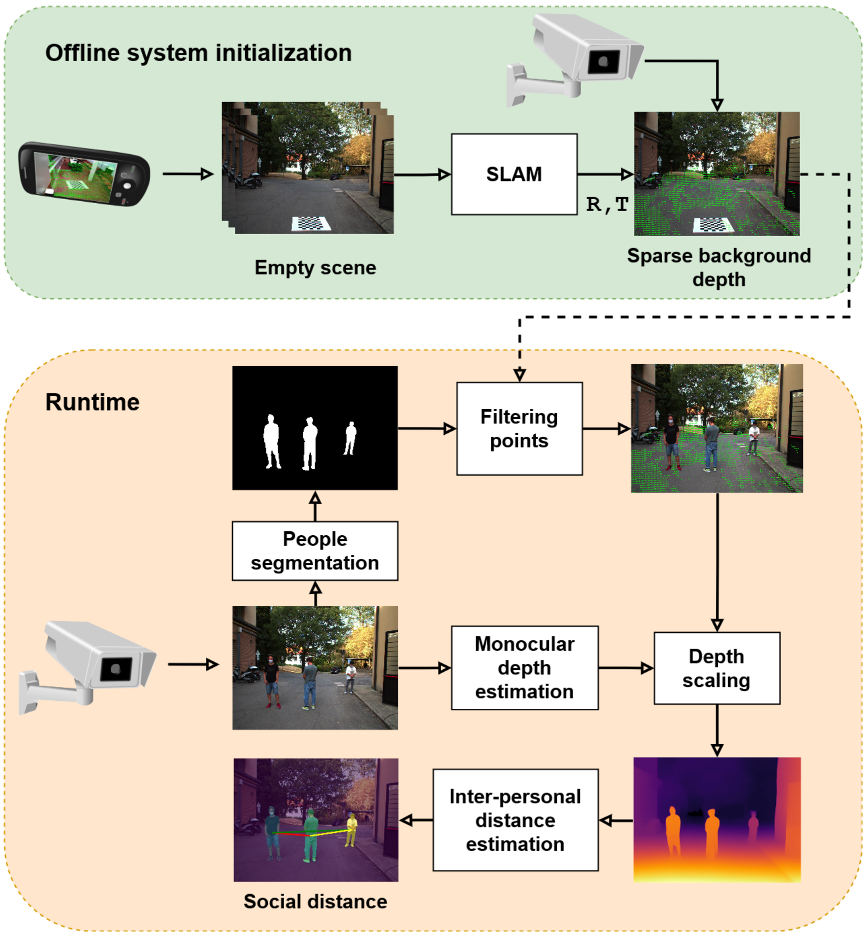


FIGURE 12.2: **Illustration of the proposed pipeline.** In an offline calibration phase (green box), we source control points in the environment employing off-the-shelf strategies. Then, when a new frame is available at runtime (orange box), we rescale the depth provided by a monocular network thanks to the available control points in the background and segment people using an instance segmentation network. Once each person's depth and mask in the scene are predicted, we can estimate the inter-personal distance.

the framework provides a metric estimation of the distance between people using only the fixed camera. The complete working pipeline is described in Figure 12.2.

12.2.1 Offline system initialization

In the initialization phase, executed only once during the first installation, a straightforward calibration procedure is needed to obtain a sparse 3D structure of the scene. This task can be accomplished in a few seconds using any device capable of inferring depth at a reasonably known scale, such as an active sensor, a stereo vision system or other techniques like SLAM. To minimize as much as possible installation requirements, in our experiments, we accomplish this task relying on the SLAM capability provided by the ARCore framework for augmented reality available for Android devices. Nonetheless, using the ARKit framework for iOS would be equivalent. These frameworks rely on an Inertial Measurement Unit (IMU) measurements and at least a single image stream to infer a sparse map of the sensed environment at a known scale. Moreover, it is worth noticing that the control points could be sourced even with more accurate and expensive devices – such as LiDARs – or with different strategies such as a full-featured SLAM system like ORB-SLAM3 [39] coupled with IMU measurements. Finally, we point out that augmented reality frameworks can seamlessly take advantage of additional setups like stereo or active sensors often available in smartphones or tablets. Nonetheless, we stick to the most constrained setup using a single camera for our experiments to reduce as much as possible the installation requirements. Specifically, the *Raw Depth API* [121] provided by ARCore enables us to obtain the sparse 3D structure of the target area within a range of [0-8] meters, providing also a confidence estimation for such data.

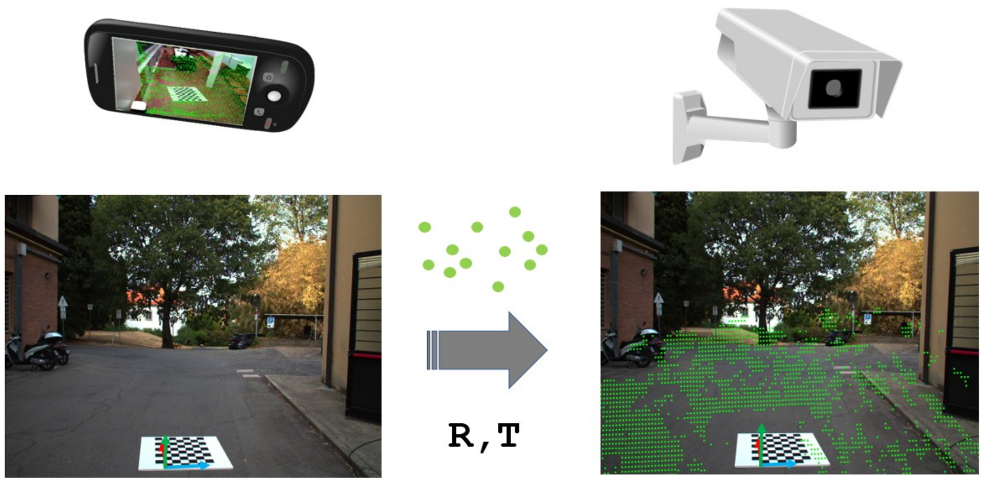


FIGURE 12.3: **System initialization.** The point cloud inferred by the smartphone on the left is remapped into the fixed camera reference system on the right by knowing the relative camera pose R,T between the two devices. Although four coplanar yet not collinear points at a known relative position would suffice to accomplish this task, for better clarity, we use in this figure a chessboard.

As depicted in Figure 12.3, at system installation, the operator selects or places at least four coplanar yet not collinear points in the sensed scene at a known relative position and records the scene moving a smartphone or a tablet in the target (although not strictly required) empty area. Then, leveraging the position of such points in both the mobile camera image and in the fixed camera, it is possible to infer the relative position between them by solving the *Perspective-n-Point* problem [176]. This procedure can be either automated using a well-known pattern such as the chessboard in Figure 12.4 or manual selecting at least four known points in the scene. It is worth observing that this procedure can be carried out even by a not-specialized operator with a consumer smartphone through a simple guided step-by-step wizard. Once the camera pose R, T between the two cameras is known, we can move depth measurements D_{SLAM} into the fixed camera reference system obtaining D_{cam} as

$$D_{\text{cam}} \sim (R|T)D_{\text{SLAM}} \tag{12.1}$$

The newly obtained depth values D_{cam} are projected from p_{SLAM} coordinates in the mobile device camera to p_{cam} pixel coordinates in the fixed camera view

$$p_{\text{cam}} \sim K(R|T)D_{\text{SLAM}}K^{-1}p_{\text{SLAM}} \tag{12.2}$$

and used as control points to perform the depth rescaling at runtime. In any case, we exploit the pixel-wise native confidence estimation of ARCore to filter out the unreliable depth values, selecting only the control points with the maximum confidence score. Figure 12.4 shows an example of control point projection over the fixed camera. Finally, it is worth noticing that control points must be sourced on fixed objects that would be available (if not occluded) even at test time, such as points on walls, furniture or trees. So, the sensed area should not contain people in the initialization phase and keep the same 3D background structure at execution time.

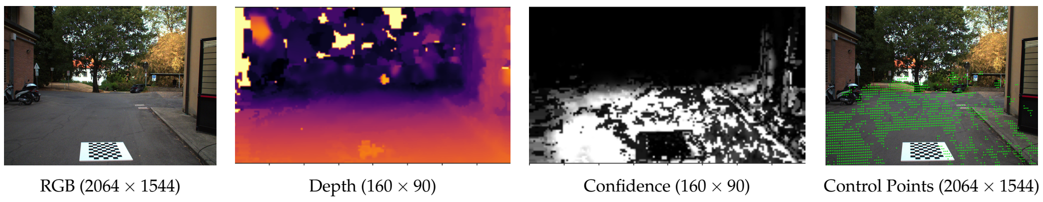


FIGURE 12.4: **Example of control points sourcing using ARCore.** We adopt the off-the-shelf SLAM framework provided by ARCore on Android devices to source control points. The estimated depth is filtered by exploiting the native pixel-wise confidence score (higher the confidence, lighter the color) provided by the same framework. Notice that the depth map generated by ARCore is way smaller than the fixed camera image, as well as of the image acquired by the smartphone. Thus, at the end of the projection onto the image of the fixed camera, only just a few sparse measures are available as control points.

12.2.2 People segmentation

For each new image acquired by the fixed camera, we identify at first people silhouettes through an instance segmentation network. For this purpose, we rely on YolactEdge [168], a fast and lightweight instance segmentation network. Specifically, starting from the pre-trained model made available by the authors, we specialize it for people through a fine-tuning for 24 epochs on a subset of the COCO 2017 dataset [166] with images mainly containing people. The silhouettes obtained by the segmentation network are used for two purposes: i) detect people to compute their inter-personal distances and ii) remove from the set of depth points inferred in the initialization stage those occluded by people, in order to scale the output of the monocular depth estimation network appropriately. Concerning the latter point, the underlying assumption is that the 3D structure of the background, where the 3D ground control points lay, does not change at runtime. To this aim, we remove people after the segmentation step. It is worth observing that, in case of considerable modifications to the underlying 3D background structure of the scene with respect to the one acquired during the system setup, a new initialization would be necessary. This requirement might occur in two extreme cases: either when most of the 3D structure of the background or the camera position has changed. In these cases, a whole initialization phase requires just a few minutes.

12.2.3 Monocular depth perception and scaling

For the reasons previously stated, we decided to rely on pre-trained networks for depth estimation without performing any fine-tuning in the target domain. Consequently, we focus our attention on networks capable of generalizing well to heterogeneous environments thanks to supervised training on large collections of data.

Although it does not allow us to retrieve scaled maps from the raw output of the monocular depth network, this strategy avoids cumbersome and time-consuming data collection required to fine-tune the network in each target domain. Moreover, it seamlessly allows replacing the depth estimation backbone with improved/newly released ones. Specifically, in our experiments we rely on the official weights released by the authors for MiDaS, DPT and PyD-Net monocular depth estimation networks, already presented in Chapters 3.3 and 5. For PyD-Net we adopt the variant trained to be robust in the wild described in Chapter 7.

As mentioned before, these networks provide only relative depth measurements and thus a scaling process is mandatory to obtain metric distances. Purposely, we rely on the sparse background depth points acquired in the initialization stage and filtered as previously outlined. We exploit such points to scale the output of the network at each frame as follows. Usually, monocular depth networks output the relative inverse depth for each pixel of the image; thus is mandatory to bring the sparse background control points into such domain. Given such background depth points, obtained as detailed in section 12.2.1, and the corresponding relative inverse depth points inferred by the monocular depth network, we pre-compute the inverse depth of the former and then, assuming a linear data distribution [245], we find the best fit between such correspondences through a linear regression to obtain the best scale α and shift β. To perform the linear regression task, we use the least-squares method as in [245]. Specifically, considering each relative inverse depth point x_i with a valid inverse metric correspondence y_i available, we obtain two scalars α and β as follows, where n is the total number of available depth points with a valid correspondence:

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \end{bmatrix} \tag{12.3}$$

Then, we scale according to $y = \alpha x + \beta$ the whole relative inverse depth map inferred initially by the network. Finally, we move back to the depth domain to obtain meaningful metric distances.

It is worth noticing that theoretically, only a set of two correspondences would suffice to obtain the α and β to scale the relative depth map. Nonetheless, using more points improves robustness to outliers in at least one of the two sets. Since the output of the monocular depth network is entirely dense, the number of points for the regression task coincides with those acquired in the initialization stage, typically a few thousand, surviving the previous phase.

12.2.4 Computing inter-personal distance

Given the dense scaled depth map and the silhouette of each person obtained through an instance segmentation network such as [329], the final step aims at computing people's distance. Although, in theory, the minimum distance between two people is the closest distance between two points belonging to each one, deploying this strategy for social distancing would be computationally expensive and prone to errors occurring primarily due to depth estimation inaccuracy. Consequently, we rely on a more robust yet approximated strategy inferring the distance between the *centroid* representing a point in space for each person in the sensed scene. Specifically, given the set of pixel coordinates assigned to a specific person Ω (defined by the

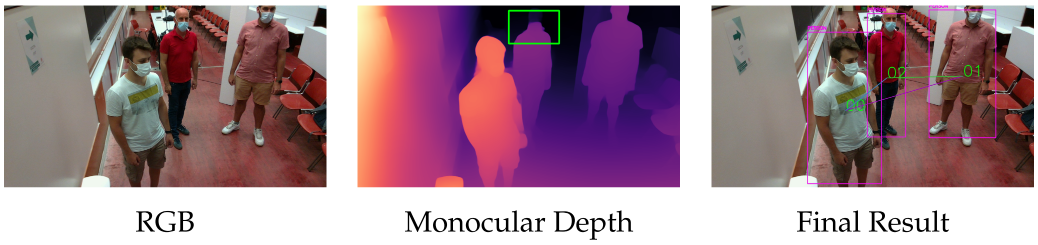


FIGURE 12.5: **Monocular failure.** Example of monocular depth failure: the head of the person is missed by the network. Nonetheless, the centroid is found on the body.

people segmentation mask obtained as described in Section 12.2.2), we compute the (u, v) coordinates of the centroid $\mathcal{C}$ in the image as follows:

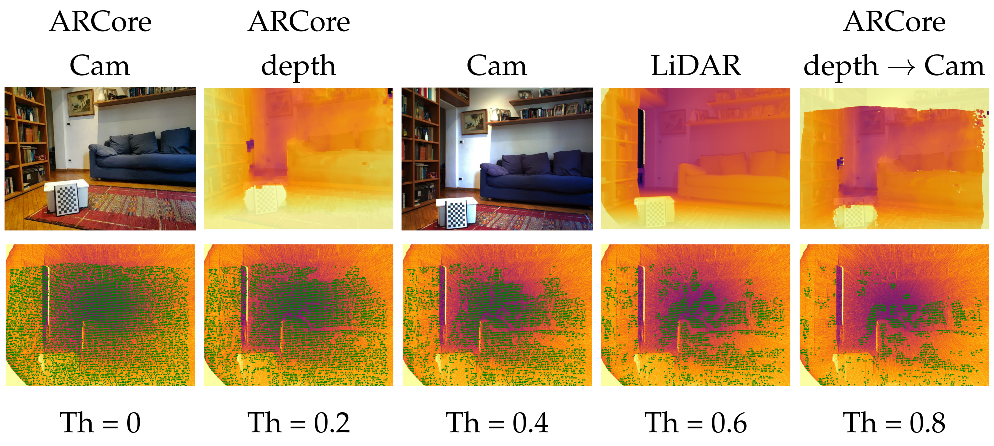
$$\begin{aligned} \mathcal{C}_u &= \tfrac{1}{|\Omega|} \textstyle\sum_{p \in \Omega} u_p \\ \mathcal{C}_v &= \tfrac{1}{|\Omega|} \textstyle\sum_{p \in \Omega} v_p \end{aligned} \tag{12.4}$$

Unfortunately, the computed $\mathcal{C}$ may not match with any of the points in Ω (e.g. due to occlusions, like a person sitting behind a bench). In such cases, we use the closest point contained in Ω to $\mathcal{C}$ as the centroid. The depth of $\mathcal{C}$ is sourced from the scaled depth map, allowing us to back-project $\mathcal{C}$ in the 3D space. Finally, we obtain the inter-personal distance as the Euclidean distance between each pair of centroids.

This strategy, even if simple, is reliable against potential errors in monocular depth estimation. Specifically, as depicted in Figure 12.5, some portions of a person, such as the head, are critical and challenging to predict correctly for many current monocular networks. In contrast, the segmentation network identifies them reliably. Thus, to increase robustness, our strategy segments the person in the 2D space and determines the 3D position of its centroid.

12.3 Experimental results

In the following section, we aim at evaluating the proposed strategy in real environments. Since open-source data sets with accurate depth values for testing purposes are not available, we collect our own set of images coupled with depth information to assess the performance of the proposed approach in indoor and outdoor settings. Specifically, we rely on an asset composed by a Livox Mid-70 LiDAR sensor and an RGB camera (FLIR BFS-U3-32S4C-C) previously calibrated and registered using a proper pattern to obtain their relative pose. It is worth noticing that this further calibration is needed only to prepare our evaluation benchmark and not for system deployment – which requires only the offline calibration presented in Section 12.2.1. Moreover, the control points are acquired using a Google Pixel2 XL and a OnePlus 6 smartphone to stress that a standard commercial device suffices in setting up the system. The point cloud obtained from the LiDAR is projected over the RGB frame leading to about 6000 depth points. However, leveraging the non-repetitive scanning technology of this specific LiDAR sensor, we can accumulate multiple point clouds of the same static scene to obtain much denser depth maps. Assuming a static scene for N consecutive LiDAR acquisitions, we can collect about 120000 points for each RGB-D example. The number of frames N is fixed to 20, and the RGB image linked to the depth data is the last one acquired by the camera. All the devices involved are not synchronized, but this is not an issue due to the constraints imposed



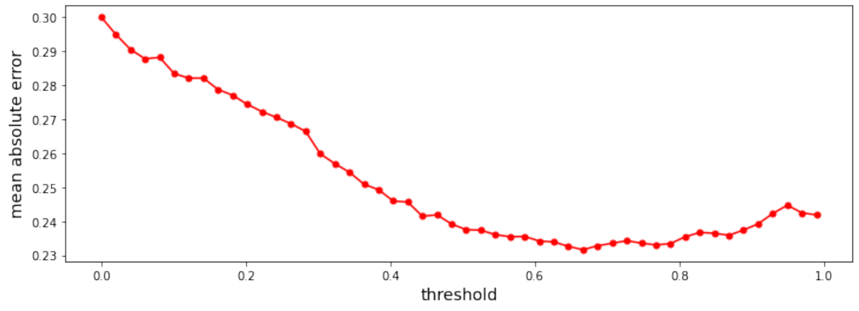


FIGURE 12.6: **Evaluation of depth data inferred with the ARCore raw depth APIs.** The first row depicts, from left to right: the image acquired with the smartphone, the depth map inferred with ARCore, the image acquired by the static camera, the depth map acquired by the LiDAR registered with the static camera, the ARCore points projected onto the fixed camera point of view. In the first row, for visualization purposes, we dilate the sparse ARCore and LiDAR depth maps to densify them and match the highest cameras resolution. From the point of view of the static camera, the second row shows the LiDAR points and the sparse points (green) obtained through ARCore for different confidence thresholds, from 0.0 to 0.8. The graph at the bottom reports the MAE of ARCore depth point wrt the LiDAR points for confidence threshold ranging from 0.0 to 1.0.

by the previously described accumulation process, which can deliver high-quality depth maps used as a reference.

In the remainder, we first evaluate the accuracy of the depth points acquired through the SLAM module available in ARCore. Then, we describe the sequences acquired to assess the effectiveness of our strategy. Lastly, according to different metrics, we evaluate the effectiveness of monitoring social distancing employing the output of monocular depth networks scaled using background control points as outlined in our proposal. As mentioned before, we consider state-of-the-art monocular depth estimation networks capable of generalizing to different and unpredictable environments without requiring additional fine-tuning in the target scene. Consequently, we include the following pre-trained models: MiDaS and MiDaS small [245], DPT and DPT-Hybrid [244], and the robust PyD-Net [230] described in Chapter 7.

12.3.1 Evaluation of control points accuracy

Since our proposal relies on the background control points to scale the depth maps inferred by monocular networks, we start by evaluating the accuracy of the depth

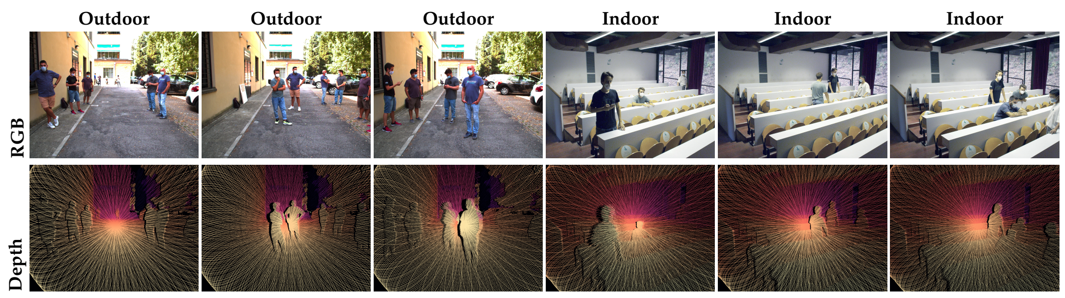


FIGURE 12.7: **Examples from the collected dataset.** In the first row, we report three outdoor and three indoor images framing still people. The corresponding LiDAR scans obtained through a Livox Mid-70, accumulating depth points for 20 frames, are reported in the second row.

obtained by the SLAM module available in ARCore. For this purpose, we scanned the indoor environment depicted in Figure 12.6 with a OnePlus 6 smartphone to compute the ARCore depth map (we report the image acquired and the corresponding depth map in the two leftmost positions at the top of the figure). Moreover, we also framed the same scene with the previously described RGB-D asset but accumulating about 500 frames from the LiDAR sensor leading to a sparse depth map with a 95% density to better match the control points. The static camera and the LiDAR dense depth map are shown in the third and fourth positions of the first row of the Figure 12.6. Then, we determined the pose of the smartphone wrt the fixed camera frame and projected the ARCore depth points onto the latter. As shown in the rightmost figure at the top, we can notice how the 3D structure of the scene inferred by ARCore is only partially overlapping with the actual image content, pointing out not slight inconsistencies in the depth map. Its amount and impact on the scaling process will be analyzed more in detail next.

In the second row, we show the LiDAR depth map from the fixed camera point of view and, filtering by different ARCore confidence threshold levels, the points inferred by ARCore in green. We can notice that even not applying the confidence filtering (th=0), the number of points inferrable with ARCore (bounded to 160×90) is much lower than those of the LiDAR but accumulating multiple scans of a static scene. However, without such accumulation, the number of points provided by the Livox LiDAR in a single scan would be much lower (about 6000). Not unexpectedly, we can notice that the higher the confidence, the lower the number of ARCore points surviving. Nonetheless, more interestingly, even setting a high confidence threshold (eg, th=0.8), the number of estimated confidence points is not negligible and quite spread across the whole scene in this image. Finally, by looking at the graph at the bottom of Figure 12.6 where we plot the MAE between the overlapping ARCore and LiDAR depth points for different threshold levels, we can observe that the error range from 30 to 23 cm in this same scene. Such a magnitude is comparable to other measurements carried out on other scenes. Moreover, we can also notice that the error decreases almost monotonically until the confidence threshold is around 0.6 and then oscillates by about 1 cm. In other evaluations, we noticed a similar behavior but better accuracy by increasing the threshold further. From this analysis and the fact that only a few yet reliable points are needed to perform the scaling process, assuming that we cannot perform this evaluation in a practical application, we conservatively set the highest threshold of 1.0 in the experiments reported in the subsequent experiments.

12.3.2 Dataset

We collected two sequences, framing indoor and outdoor domains, for a total of 83 frames with accurate depth labels. Figure 12.7 depicts frames sampled from each sequence.

In the following, we provide additional details for each collected sequence. We refer to control points as those scene points with an associated depth value obtained through the ARCore SLAM framework, which has a confidence score, estimated by ARCore, equal to one.

Outdoor sequence The scene, counting 46 frames, depicts six people walking in a driveway with a poorly textured yellow wall at the left and parked cars at the right. The scene contains 838 control points, while the average distance of the people from the camera is 5.15 m.

Indoor sequence This sequence, counting 37 frames, simulates the presence of three to four people in a university classroom. We found about 679 control points with SLAM in the scene, while the average distance from the camera is 4.85 m.

12.3.3 Inter-Personal distance evaluation

For each pair of people in each test frame, we first compute the inter-personal distance using the proposed strategy explained in Section 12.2.4. Then, we measure the error between the latter and the one measured using the reference depth map obtained by the LiDAR sensor, showing how much our approach deviates from the metric measurements enabled by an accurate depth sensor. Table 12.1 illustrates the results of this evaluation, grouping outdoor and indoor environments, respectively. Specifically, to assess the quality of our approach, we report the results scaling the depth maps using three different types of control points: (a) depth inferred by a monocular SLAM system as outlined in our proposal, (b) the raw depth inferred by the LiDAR and (c) the raw depth inferred by the LiDAR constrained to a depth range similar to that of ARCore (not farther than about 10 meters).

Focusing on (a), we can notice how, in general, the deviation ranges from 0.27 cm (DPT-Hybrid), comparable with the accuracy of the ARCore control points, to a much higher value of more than 1 meter (MiDaS small). It is often related to the complexity of the depth network deployed, with more complex ones more accurate. Although this is not always true, on average, MiDaS and DPT perform overall better than others, with the lightweight PyD-Net model sometimes really close or better in the outdoor environment. By comparing the performance in the two scenarios, we can notice how only MiDaS can yield an accuracy in the inter-personal distance below half a meter considering any range between people. Limiting such an evaluation only for people at a distance below 3 meters, excluding one case (PyD-Net in Indoor), all the networks improve their performance by a significant margin with MiDaS and DPT models more prominently than others. This latter evaluation is particularly important for social distance monitoring since potential violations occur when people get closer. According to the evaluation reported in the table, MiDaS is the most effective network yielding an uncertainty of 33 cm on average.

Concerning results (b) and (c) in the table, it is interesting to analyze the framework behavior when providing more accurate control points. Indeed the ARCore depth accuracy is far from being perfect as yet observed in Section 12.3.1. In configurations (b) and (c), although not consistently, most networks improve their performance. Sometimes the improvement is notable and higher when the sparse background data are closer, such as for PyD-Net yielding results almost overlapping

	Any range		<3 meters	
Model	Outdoor	Indoor	Outdoor	Indoor
PyD-Net	0.44	0.76	0.33	0.82
MiDaS	0.58	0.42	0.39	0.27
MiDaS small	0.69	1.09	0.52	0.97
DPT-Hybrid	0.44	0.73	0.27	0.46
DPT	0.48	0.65	0.31	0.39

(a) ARCore points

	Any range		<3 meters	
Model	Outdoor	Indoor	Outdoor	Indoor
PyD-Net	0.58	0.51	0.47	0.34
MiDaS	0.54	0.43	0.32	0.32
MiDaS small	0.77	0.62	0.67	0.49
DPT-Hybrid	0.29	0.39	0.24	0.23
DPT	0.33	0.44	0.24	0.30

(b) Livox points

	Any range		<3 meters	
Model	Outdoor	Indoor	Outdoor	Indoor
PyD-Net	0.42	0.53	0.33	0.34
MiDaS	0.44	0.43	0.35	0.31
MiDaS small	0.60	0.61	0.53	0.48
DPT-Hybrid	0.34	0.40	0.25	0.23
DPT	0.37	0.44	0.31	0.30

(c) Livox points (<10 meters)

TABLE 12.1: **Evaluation of Inter-Personal distances.** system scaled with (a) ARCore points using confidence threshold 1, (b) unconstrained LiDAR measurements, and (c) LiDAR measurements below 10 meters. The inter-personal distance predictions are compared to the reference depth provided by the LiDAR, averaged over all pairs and images. We report results in terms of MAE for the indoor and outdoor sequences, considering unconstrained inter-personal distances and below 3 meters.

with the DPT networks as reported in (C) for distances smaller than 3 m. With both LiDAR data distributions, the best accuracy considering distance below 3 m gets

	Outdoor								
	Safe			Risky			Dangerous		
Method	P	R	F	P	R	F	P	R	F
PyD-Net	0.94	0.83	0.88	0.69	0.79	0.74	0.45	0.70	0.55
MiDaS	0.91	0.87	0.89	0.73	0.70	0.71	0.47	0.75	0.58
MiDaS small	0.88	0.85	0.86	0.70	0.63	0.67	0.31	0.68	0.43
DPT-Hybrid	0.94	**0.89**	0.91	**0.79**	0.80	0.79	**0.58**	0.91	**0.70**
DPT	**0.95**	**0.89**	**0.92**	**0.79**	**0.81**	**0.80**	0.51	**0.92**	0.66
(a) Homography	0.90	0.97	**0.93**	**0.89**	0.77	**0.83**	**0.65**	0.58	0.61
(b) Aghaei et al. [5]	0.35	**0.98**	0.51	✗	✗	✗	✗	✗	✗

	Indoor								
	Safe			Risky			Dangerous		
Method	P	R	F	P	R	F	P	R	F
PyD-Net	0.53	0.81	0.64	0.47	0.36	0.41	0.83	0.28	0.42
MiDaS	0.79	0.79	0.63	0.63	0.72	0.67	0.67	0.44	0.53
MiDaS small	0.59	**0.89**	**0.71**	**0.68**	0.46	0.55	**0.88**	0.47	0.61
DPT-Hybrid	**1.00**	0.43	0.60	0.55	0.61	0.58	0.46	**1.00**	0.63
DPT	**1.00**	0.45	0.62	0.60	**0.84**	**0.70**	0.67	0.84	**0.74**
(a) Homography	✗	✗	✗	✗	✗	✗	✗	✗	✗
(b) Aghaei et al. [5]	✗	✗	✗	✗	✗	✗	✗	✗	✗

TABLE 12.2: **Evaluation about risk detection.** Evaluation of pair-wise inter-personal distance estimation for predicting Safe, Risky and Dangerous situations with respect to the reference depth provided by the LiDAR, averaged over all pairs and images. For each sequence, we highlight with **bold** the best result within our methods and with **underline and bold** the best result overall. The tag ✗ means that the information is not available, either because the method cannot be applied at all due to the scene's constraints or it does not provide the desired metric.

close to 20 cm, slightly better than deploying ARCore depth data. On the one hand, this fact highlights that the uncertainty of depth measurements needed to scale the network has a not marginal impact. On the other hand, most of the depth uncertainty occurs due to the ill-posedness problem the monocular depth networks face, although some are undoubtedly more effective than others.

12.3.4 Detecting violations

In the previous experiment, we validated our framework by computing the average error, in meters, between the expected and predicted inter-personal distance among people. However, another critical question is: *how many times the system fails in predicting violations of the necessary inter-personal distance?* This point is crucial because even if the depth prediction is partially wrong, social distancing could be effectively fulfilled. Moreover, the concept of risk depends on the context: distances lower than one meter could be crucial for some critical applications, but a too strict constraint in less critical ones. For this reason, we evaluate next the performance of the system with three different classes of risk, Safe, Risky and Dangerous, representing when the relative distance among two people is > 2 m, in the range $[1, 2]$ m and < 1 m respectively. We evaluate the Precision (P), the Recall (R) and the F1 score (F), in

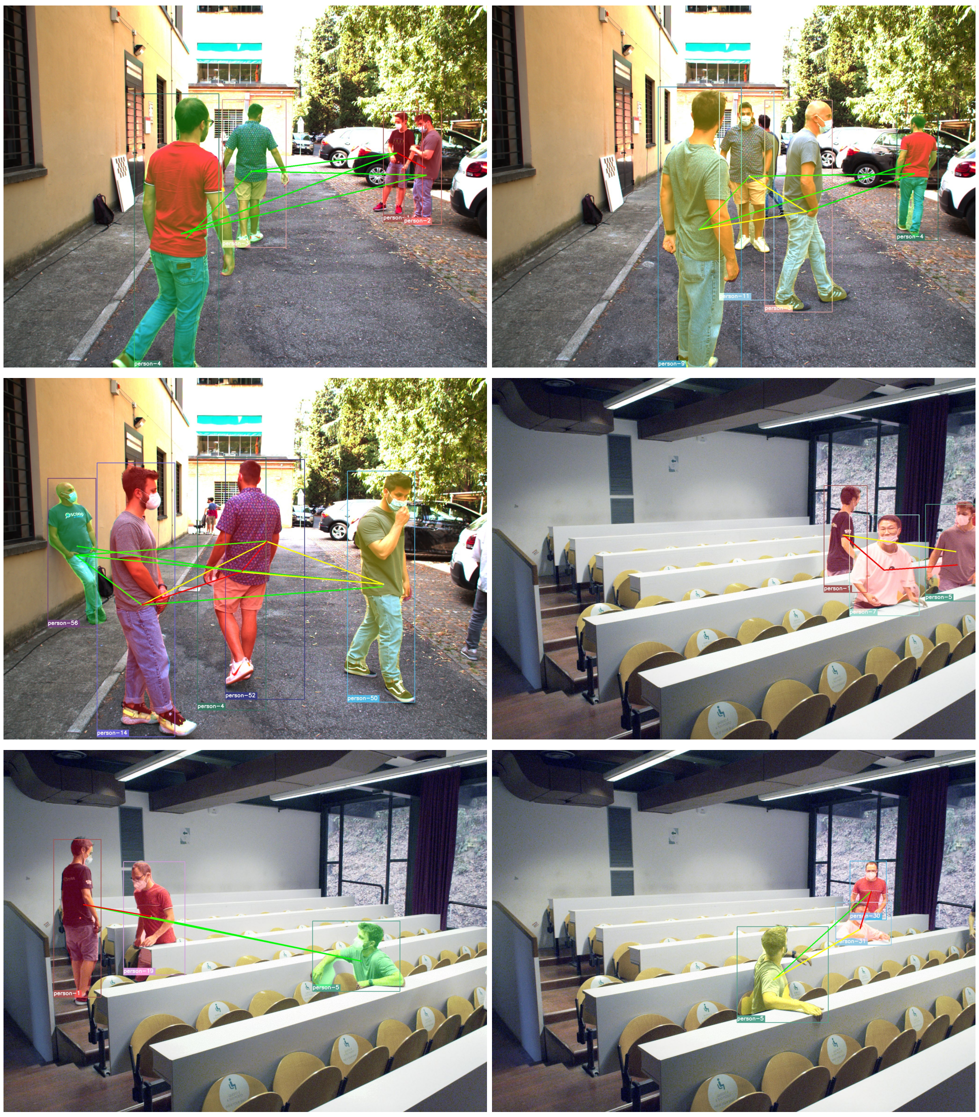


FIGURE 12.8: **Qualitative results.** Examples from the collected scenes. The computed inter-personal distance is superimposed with a color overlay to the images – green, yellow and red, mean safe , risky and dangerous .

$[0, 1]$ format, of all the depth models using these three classes of risk. Table 12.2 collects the outcome of this evaluation, on both outdoor and indoor data. We compare the various flavours of our approach to two ways to recover interpersonal distances by means of homography computation (a) finding the position of each person as the center of the side of the bounding box touching the floor (e.g. [66, 74]) or (b) applying the method by Aghaei et al. [5], automatically detecting inter-personal distances lower than 2m.

Starting from outdoor samples, we notice how all models effectively distinguish safe situations, reaching a precision often superior to 0.90 and a recall between 0.83 and 0.89. DPT models allow for the best results, reaching in the worst case a 0.91 F1 score. MiDaS and PyD-Net perform similarly, reaching an F1 score slightly below 0.90. Excluding MiDaS small, the precision results are better than exploiting the homography (a) and vastly outperforms [5] (b), while always resulting worse in terms of recall. The F1 score achieved by our solution is almost equivalent, using DPT models, with (a), and much better than (b) [5]. The latter yields many false positives, often identifying violations of the social distancing constraint even when not occurring.

Analyzing risky situations, the precision for all networks drops at about 0.79 for DPT models and around 0.70 for the others. DPT models and MiDaS keep a similar score regarding the recall metric, while PyD-Net performs better and MiDaS small drops to 0.63. DPT variants achieve F1 around 0.80, still performing close to (a), achieving higher precision. The framework (b) by Aghaei et al. [5] is implemented to distinguish violations of the 2m distances, not allowing for finer processing and thus not able to detect risky and dangerous situations.

Finally, all models' precision scores are generally significantly lower concerning dangerous situations, whereas recall metrics are higher, except PyD-Net than those observed for risky situations. (a) results are better in terms of precision, worse concerning the recall. The best F1 score overall is achieved by DPT variants.

Moving to the indoor scene, it is worth noticing how methods based on homography (a) and (b) cannot be computed here, as we can notice by looking at Fig. 12.7 since in this specific environment the ground plane is not visible. Nevertheless, our method can still reliably monitor social distancing since it does not require the ground plane to run. We observe excellent precision scores for DPT models when dealing with safe cases, while other models are significantly less accurate. The opposite behaviour occurs for the recall. Surprisingly, the best F1 score 0.71 is achieved by MiDaS small, with MiDaS and PyD-Net following. Concerning risky conditions, in most cases, the performance degrades significantly, especially for lightweight models MiDaS small and PyD-Net. Finally, we can notice mixed results concerning dangerous situations with DPT models achieving the best results for recall and F1 metric, but outperformed by MiDaS small in precision. Looking at the recall, we can notice PyD-Net struggling, as for the risky case.

In summary, the evaluation with the three classes of risk highlights that the models are pretty effective in handling safe or risky situations between people, especially in the former case. Nonetheless, they become more unreliable when facing dangerous situations with distances falling under one meter. DPT models achieve the best performance in most cases while lightweight models, PyD-Net and MiDaS small, further emphasize the outlined behavior. In general, our approach is competitive compared to homography computation (a) and much better than (b) [5] overcoming their main limitation – the need for the ground plane to be visible and detectable. We report some qualitative results concerning the outcome of the proposed framework in Figure 12.8.

12.3.5 Runtime analysis

Table 12.3 reports the performance of each component of the proposed overall framework (as average time per frame) on CPU and GPU. Moreover, we also report the same metric for the two competitors on the same hardware when feasible. The depth estimation models vary from more than 4 (DPT) to more than 60 FPS (PyD-Net and MiDaS small) on a quite outdated Nvidia GTX Titan X GPU. The YolactEdge instance segmentation network accounts for 0.055s on the same hardware. Focusing on the overall system, when using the same GPU, our approaches run at frame rate ranging from 9.4 FPS, with PyD-Net and MiDaS small, to more than 3, with DPT, comparable to competitors. With a pure CPU system, the runtime of the instance segmentation network becomes more prominent, especially compared to the fastest depth estimation networks. Nonetheless, the largest depth estimation models, such

Method	Component	CPU	GPU
ours/	PyD-Net	0.033s	0.016s
	MiDaS small	0.067s	0.016s
	DPT	2.125s	0.225s
	YolactEdge	0.349s	0.055s
	Remaining computations	0.050s	0.035s
Homography-based	YolactEdge	0.349s	0.055s
	Remaining computations	0.050s	0.035s
Aghaei et al. [5]	Social-Distancing	-[1]	0.110s

TABLE 12.3: **Runtime evaluation.** We measure the performace of each subsystem when adopting different depth models coupled with two different instance segmentation networks at resolution 826x618. The CPU and GPU employed are an Intel Core i7-7700K and an Nvidia GTX Titan X respectively.

as DPT, account for even higher execution time. Our framework runs at a frame rate ranging from more than 2, with MiDaS small and PyD-Net – slightly slower than the homography-based approach, to 0.4, with DPT. Finally, we conclude by reporting that with an NVidia Jetson Nano, our framework runs at about 0.5 FPS using the fastest configurations, with power consumption below 10W, suggesting the practical deployment of our approach even on low-power embedded systems is around the corner.

12.4 Limitations

The proposed system proved to be effective in monitoring inter-personal distances in non-critical applications. Although versatile and effective even when the ground plane of the scene is not visible or there are multiple planes, like when sensing a staircase, it has some limitations and might fail under certain conditions. In the following, we summarize the most notable ones.

Monocular generalization. The monocular network itself represents a critical aspect. Even if we adopt networks with strong generalization capabilities, they might fail when misleading images, such as those acquired by unusual perspectives. In this case, the network could fail to predict the correct depth for the scene, and the scaling alone would not recover its correct geometry. However, as discussed in Section 12.2.4, since we compute the centroids in the 2D space, partial failures in monocular predictions are tolerable for inter-personal distance computation as long as the depth of the centroid is correct. Even if the monocular network assigns a wrong depth to the head of the person (Monocular Depth image in Figure 12.5), the centroid is found on the body, where the depth is consistent.

The problem outlined is well-known in literature, and in future works, it could be tackled employing learned refinement strategies [346] or performing in-domain fine-tuning. The choice of using the centroid as defined in Section 12.2.4 may not be the best when searching for the social distance, although most existing approaches follow similar strategies [66, 5]. A stricter approach could be to use the nearest points

[1]We could not install the CPU version of OpenPose on our hardware. Nevertheless, the authors claim that the default CPU version takes ∼0.2 images per second on Ubuntu (∼50x slower than GPU) while the MKL version provides a roughly 2x speedup at ∼0.4 images per second.

between each pair of people. However, this method is computationally expensive and less robust to noise in depth computation. Future work could concern finding a proper way to retain robust measures even when deploying more complex approaches for people relative localization.

Control points sourcing. Another issue concerns the control points needed to scale the predicted maps. A subset of at least two points must be visible at inference time. Thus, an extremely crowded environment might potentially result in a scene without visible control points. However, this seems very unlikely, and we never faced such situations in our experiments. Additionally, control points could be detected on objects that might move, e.g. cars in Figures 12.7 and 12.8. However, these points could be easily discarded at test time if properly detected, for instance, employing background subtraction techniques.

People detection. Lastly, another source of errors is the people segmentation network. When the network fails in segmenting different people or, in the worst case, it misses a person; our framework would predict wrong inter-personal distances. Nonetheless, we would like to point out that this problem is shared with other vision-based social preserving applications and that it could be partially alleviated by exploiting temporal information from the video stream (e.g. using the Kalman filter to forecast the next position of the centroid).

12.5 Conclusions

In this chapter, we proposed a framework aimed at social distance monitoring from a single image. The system requires a single, static RGB camera, making it suitable for already deployed infrastructures such as surveillance cameras. In contrast to most other approaches based on a single camera proposed in the literature, our framework is robust even when the ground plane is not visible, or there are multiple ones, such as in the presence of staircases. To retrieve the metric scale factor out of the output of monocular depth networks, we propose a simple yet effective strategy to source sparse depth points requiring only a consumer smartphone. Experimental results highlight that our proposal is effective, particularly as a versatile and cheap solution for non-critical applications.

Chapter 13

Closing remarks

In this work, we discussed depth estimation, with a particular focus on the monocular case: even if inferring the depth from a single image of the scene is challenging, solutions would have countless applications in real use-cases. In Chapter 4 we presented a novel network, MonoResMatch, that resembles a stereo model: the initial monocular depth is used to warp the reference features, obtaining a virtual stereo pair; then, the two feature volumes are used to refine the disparity using a correlation layer. Following previous literature, we train the model without the need of ground-truth labels but leveraging stereo pairs. Nonetheless, instead of adopting image reconstruction as the main loss for optimization, we exploit classical stereo methods to distill proxy labels. These labels prove to be effective and can provide a stronger signal at training time than self-supervision. MonoResMatch tries to achieve the best accuracy and does not take into account computational costs. However, as we discussed, it is not always possible to deploy cumbersome and power-hungry GPUs, especially in the case of mobile and portable devices. Chapter 5 presented a family of lightweight models, called PyD-Net, able to run also on compact and low-power devices: the peculiar design allows for stopping the execution at a lower resolution, preserving time and resources when needed. Then, we pointed out that, sometimes, we are also interested in obtaining additional information about the scene in addition to depth. In fact, depth is crucial but could not be enough to implement applications that go beyond geometry. A popular example is self-driving cars, in which the motion of objects between frames and semantic labels are crucial information. In Chapter 6 we presented a single, lightweight network for comprehensive scene understanding from monocular videos: the ΩNet model can predict depth, optical flow and semantic labels exploiting the correlations among the tasks. These cues about the scene can be mixed to compute, for instance, the mask of moving objects in the scene. Afterwards, we dealt with the generalization problem of monocular models. A requirement towards real use-cases at scale for monocular applications is the reliability in the wild, but popular training strategies, both supervised and self-supervised, exploit small or well-contextualized datasets that prevent it. Recent methods as [245] leverage millions of different data for training purposes, but processing this large amount of images requires time and resources. In Chapter 7 we showed that a simple and fast teaching-student paradigm allows infusing the knowledge of off-the-shelf models into both large and small models. These models can be deployed and used in real applications. Furthermore, the availability of off-the-shelf models with notable generalization capabilities paves the way towards different uses other than simply measuring distances. In Chapter 8 we illustrated how to generate thousands of ground-truth optical flow maps starting from a dataset of still images and an off-the-shelf monocular depth model. Sourcing optical flow labels is expensive and requires a great effort, but the proposed pipeline breaks down the costs and produces labels for images with real texture. Chapter 9 showed that monocular models are beneficial also in improving the performance of deep stereo models: given an unlabeled stereo dataset for training, we first obtain a peculiar monocular model supervised by classical stereo methods, then we rely on this expert monocular model to guide the training of a binocular stereo network. This strategy proves to be more effective than the conventional self-supervised approach, leading to stereo models with good performance even in presence of occlusions. We then focused on detecting unreliable values in monocular predictions, because in many cases our systems are reliable against some missing values in predictions, while errors in measurements may have dramatic effects. Chapter 10 studied different strategies for uncertainty estimation for self-supervised monocular models, and we proposed a *self-teaching* paradigm beneficial for the task. In Chapter 11 we addressed the problem of disparity refinement: the reference image and the outcome of classical or deep stereo models are given as input to a neural refinement module, able to predict refined disparity maps at arbitrary resolution. Although the network does not perform a conventional monocular inference, the reasoning on a single image combined with the strong prior given by the input disparity allows the model to successfully detect and ameliorate errors, producing competitive refined maps for out-domain samples. Moreover the peculiar feature interpolation scheme enables to handle easily both balanced and unbalanced stereo configurations. Finally, Chapter 12 presented a real problem arduous yet crucial to address due to the global pandemic: monitoring social distancing violations. We have seen how off-the-shelf robust monocular models can be employed to detect when two or more people are too close. Differently from other strategies proposed in the literature, monocular depth requires only a single camera, meaning that we can leverage the already available infrastructure of surveillance cameras. Since these models predict depth up to a scale, we also outline a fast and low-effort strategy to obtain the missing metric information. Specifically, during a preliminary calibration phase, the technical operator leverages a consumer device to collect measures for the monitored scene; these cues about the scene to monitor are then exploited after the deployment of the system to retrieve the metric scale. Experimental results highlight that the proposed approach is not ready to be adopted in critical applications, but it can be a cheap and non-intrusive alternative in case of low-risk surveillance.

13.1 Limitations

Monocular depth is a promising technology and solutions based on deep learning seem to be ready for a broader set of applications and uses even out of the research community. Nonetheless, we have to be aware of its main limitations.

The first issue is about the scale of monocular predictions: the ambiguity of the problem forces solutions to be up-to-scale. The main training strategies we discussed in the past chapters suffer from this problem. For example, self-supervised methods that use monocular videos for training predict up-to-scale depth maps by construction because the estimated camera pose can be at an arbitrary scale, and the depth has to be consistent with it. Strategies that leverage stereo pairs for training and learn to predict disparities, whether they are self or weakly supervised, suffer this problem in generalization. These disparities do not encode the real matching between the two views of a stereo setup, because this setup is not available in practice. Instead, they represent the matching in a virtual stereo setup, i.e. they predict a left-aligned disparity map according to a plausible right. As long as the training setup is close to the testing one (i.e. we are using the same camera in a similar environment), the inferred disparity map can be representative of the real data distribution. Instead, if we apply the method to different data captured by heterogeneous cameras, the predicted disparity maps may not be meaningful. A similar behaviour occurs for supervised strategies: the scale learned by the model using depth labels (e.g. meter units) can be different (e.g. millimetres o hundreds of kilometres) at test time. Although the scale may not be an issue in some applications (as in Chapter 8), we require additional strategies and data to obtain in-scale depth measurements, like the one illustrated in Chapter 12.

This problem is strictly related to another one: the data variety. As discussed in Chapter 7, many methods exploit one or few domains for training, causing well-known domain-shift issues. Recent works as [245] are pivotal for this purpose and show that large datasets with depth (proxy) labels are crucial to tackling the complexity of the problem at scale. However, the intrinsic limitations make the challenge extremely hard to solve, if not impossible, even when increasing the knowledge of the model. For the same motivations, the accuracy achieved by state-of-the-art monocular models is not comparable to binocular stereo models or active sensors, and probably it will never be. Nonetheless, for non-critical applications, they represent a valuable, cheap and effortless alternative, as shown in Chapter 12.

A final limitation regards the intrinsic limitation of single-image monocular depth estimation strategies. Many monocular methods, such as [80, 79, 245], process single images at test time, and this represents a limitation when video sequences are available since important context information, which is shared across the time dimension, is inevitably lost. Nowadays, having video sequences is a common requirement for many applications, so single-image monocular solutions do not represent the best choice for these tasks. On the other hand, these strategies are effective when processing unpaired images, e.g. internet sampled images as in Chapters 7 and 8. Therefore, this last limitation can be easily addressed, as we are going to discuss in the next section.

13.2 Future directions

Although in this work we have described novel architectures and training strategies for monocular depth estimation, this technology is recent and thus the research must not be stopped. In the following, we briefly summarize interesting future directions.

First, modern devices, even mobile ones, are increasingly powerful. This means that the trade-off between accuracy and speed can be improved in the future without the need to sacrifice excessively in terms of model complexity. For this reason, more powerful operators, such as the Attention mechanism [311], or ad-hoc strategies for high-res image processing can be employed even for mobile applications.

Powerful devices unlock also another paramount topic, i.e. monocular depth adaptation using video sequences. In Chapter 2.5 we pointed out that adaptation represents a valuable strategy to improve the accuracy of the model using directly test time data. Of course, this is possible if our devices have enough computational capabilities, but also if we have a strong learning signal. As discussed in this work, self and weakly supervised strategies can be employed for this purpose, but the research on this direction is far from being fully explored. Particularly attractive is the combination of monocular models with structure-from-motion (SfM) and simultaneous localization and mapping (SLAM) techniques. The work of Tiwari et al. [296] represents a preliminary step towards this direction, but their self-improving loop does not run in real-time and has to be performed multiple times for best results. A valuable future direction tackles this limitation and combines the well-known geometric constraints of SLAM to guide a monocular models designed to learn quickly, e.g. by means of meta-learning techniques as in [288]. In Chapter 12 we have seen that nowadays mobiles already provide off-the-shelf SLAM pipelines and multiple sensors (e.g. stereo cameras), so this integration can be done seamlessly with the possibility to include sensor-fusion strategies.

Structure-from-motion (SfM) has been employed also in Neural Radiance Fields [197] (NeRF), a recent strategy for novel view synthesis of complex scenes. Specifically, NeRF encodes the 3D scene in a neural network and can be trained using only multiple images (e.g. 80 images) of a static scene, where the camera poses are given by SfM pipelines as COLMAP [267]. Despite the interest in improving NeRF by reducing the number of training images [348, 211], tackling dynamic scenes [239, 162] or boosting the inference time [248, 204], the combination of NeRF with monocular and binocular stereo models has not been totally addressed yet by previous works [324, 251]. The room for improvement is given by the possibility to combine with success the different strategies: on the one hand, the monocular or the stereo network provide a strong prior about each view of the scene, on the other NeRF introduces the 3D consistency that is not exploited when using single views.

A final, attractive way consists in leveraging the astonishing progress made by neural networks in different topics, such as the multi-modal architecture CLIP [240]. The ability of CLIP to link visual concepts with textual messages represents an intriguing path towards a novel generation of algorithms. The Dream Fields method of Jain et al. [124], in which NeRF is optimized from both visual and text as supervision, is an example of this new approach. Similar strategies can be beneficial also for monocular methods to tackle scene ambiguities, or to encode similarities between images at different levels for a targeted model fine-tuning or adaptation. For instance, we can imagine a system that learns with a singe, static camera exploiting the cues that a complex multi-modal model provides about the scene (e.g. by doing human-like web queries for the height of objects, already available CAD models, etc). These are only few, exciting future directions for an ever-growing field with unlimited applications.

Appendix A

Additional details for Chapter 4

A.1 Training protocol

We implemented our architecture using the TensorFlow framework, counting approximately 42.5 million of parameters, summing variables from the multi-scale feature extractor (0.51 million), the initial disparity stage (41.4 million) and the refinement module (0.6 million). In the experiments, we pre-trained MonoResMatch on Cityscapes (CS) running about 150k iteration using a batch size of 6 and random crops of size 512×256 on 1024×512 resized images from the original resolution. We used Adam optimizer [139] with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. We set the initial learning rate to 10^{-4}, manually halved after 100k and 120k steps, then continuing until convergence. After the first pre-initialisation procedure, we perform fine-tuning of the overall architecture on 22,600 KITTI raw images from the Eigen KITTI split (K). Specifically, we run 300k steps using a batch size of 6 and extracting random crops of size 640×192 from resized images at 1280×384 resolution. At this stage, we employed a learning rate of 10^{-4}, halved after 180k and 240k iterations. We fixed the hyper-parameters of the different loss components to $\alpha_{ap} = 1$, $\alpha_{ds} = 0.1$ and $\alpha_{ps} = 1$, while $n_i = 4$ and $n_r = 3$. As in [79], data augmentation procedure has been applied to both images from CS and K at training, in order to increase the robustness of the network. At test time, we post-process disparity as in [79, 237, 341]. Nevertheless, we preliminary highlight that, differently from the strategies mentioned above, effects such as disparity ramps on the left border are effectively solved by simply picking random crops on proxy disparity maps generated by SGM, as clearly visible in Figure 4.2 (c).

Proxy supervision is obtained through SGM implementation from [280], which allows us to quickly generate disparity maps aligned with the left and right images for both CS and K. We process such outputs using left-right consistency check in order to reduce the numbers of outliers, as discussed in Section 4.2 using an ϵ of 1. We assess the accuracy of our proxy generator on 200 high-quality disparity maps from KITTI 2015 training dataset, measuring 96.1% of pixels having disparity error smaller than 3. Compared to Tonioni et al.[300], we register a negligible drop in accuracy from 99.6% reported in their paper. However, we do not rely on any learning-based confidence estimator as they do [233], so we maintain label distillation detached from the need for ground-truth as well. Since SGM runs over images at full resolution while MonoResMatch inputs are resized to 1280×384 before extracting crops, we enforce a scaling factor to SGM disparities given by $\frac{1280}{W}$, where W is the original image *width*. Consequently, the depth map estimated by MonoResMatch must be properly multiplied by $\frac{W}{1280}$ at test time. The architecture is trained end-to-end on a single Titan XP GPU without any stage-wise procedure and infers depth maps in 0.16s per frame at test time, processing images at KITTI resolution (i.e., about 1280×384 to be compatible with MonoResMatch downsampling factors).

Appendix B

Additional details for Chapter 5

B.1 Training protocol

Our networks are implemented using TensorFlow. As we are using a custom encoder, ImageNet [58] pre-training is not used, differently from most recent works [80, 321], thus pre-training is conducted for 50 epochs on Cityscapes dataset [55] following [79], then the fine-tuning is carried out on the KITTI Eigen training split for 200 epochs as in [230]. Since losses computation occurs at downsampled scales along the pyramid levels, training the original PyD-Net is quite fast. On the other hand, PyD-Net2 doubles the training time since loss computation always occurs at the highest resolution. In particular, to perform 50 epochs of training PyD-Net requires 10 hours while PyD-Net2 requires 20 hours. Training of all networks was carried out on a NVIDIA Titan Xp single GPU. All models are optimized using Adam [139] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. A learning rate of 10^{-4} is used for the first 60% epochs, halved every 20% epochs until the end as in [230]. Standard data augmentation is performed as in [79], by randomly flipping input images horizontally and applying the following transformations: random gamma correction in [0.8,1.2], additive brightness in [0.5,2.0], and color shifts in [0.8,1.2] for each channel separately. No post-processing [79] is applied to halve test-time runtime and memory footprint.

Appendix C

Additional details for Chapter 6

C.1 Architectures of the networks

In this section we present the networks composing ΩNet (highlighted in red in Figure 6.2).

Depth and Semantic Network (DSNet). We build a single model, since shared reasoning about the two tasks is beneficial to both [352, 52]. To achieve real-time performance, DSNet is inspired to PyD-Net [230], already presented in Chapter 5, with several key modifications due to the different goals. We extract a pyramid of features down to $\frac{1}{32}$ resolution, estimating a first depth map at the bottom. Then, it is upsampled and concatenated with higher level features in order to build a refined depth map. We repeat this procedure up to half resolution, where two estimators predict the final depth map D_t and semantic labels S_t. These are bi-linearly upsampled to full resolution. Each conv layer is followed by batch normalization and ReLU, but the prediction layers, using reflection padding. DSNet counts 1.93 million parameters.

Camera Network (CamNet). This network estimates both camera intrinsics and poses between a target I_t and some source views $I_s (1 \leq s \leq 3, s \neq t)$. We extract a pyramid of features down to $\frac{1}{16}$ resolution for each image and concatenate them to estimate the 3 Euler angles and the 3D translation for each I_s. As in [82], we also estimate the camera intrinsics. Akin to DSNet, we use batch normalization and ReLU after each layer but for prediction layers. CamNet requires 1.77 million parameters for pose estimation and 1.02 thousand for the camera intrinsics.

Optical Flow Network (OFNet). To pursue real-time performance, we deploy a 3-frame PWC-Net [287] network as in [170], which counts 4.79 million parameters. Thanks to our novel training protocol leveraging on semantics and self-distillation, our OFNet can outperform other multi-task frameworks [15] built on the same optical flow architecture. We do not pre-train the network using synthetic data.

C.2 Losses

To train the DSNet module, we rely on a multi-task loss function based mainly on two terms. In particular, a depth term is in charge of minimize the discrepancy between the target image I_t and an image I_s, warped as $\tilde{I}_t^s$, from a monocular sequence while a semantic term is used to learn semantic labels from proxy label distilled by a pre-trained network.

Depth term. According to the self-supervised training paradigm proposed in [79], we adopt a photometric loss function consisting in a weighted combination between the Structural Dissimilarity Measure (DSSIM) and the standard $L1$ loss, as in Chapter 4.4. In addition, a per-pixel minimum strategy [80] is used to solve occlusion/disocclusion by simply picking the minimum error between each pair I_t and I_s instead of averaging them.

$$\mathcal{L}_{ap}^{D} = \sum_{p} \min_{s}(\alpha \mathcal{L}_{DSSIM}(p) + (1-\alpha)|I_t(p) - \tilde{I}_t^s(p)|) \tag{C.1}$$

where p denotes pixel coordinates, $\tilde{I}_t^s$ a source image I_s warped according to estimated depth and camera pose. In our experiments, we set $\alpha = 0.85$.

A smoothness term is also used to penalize large depth differences between adjacent pixels when depth discontinuities do not co-occur with strong RGB gradients. We presented this term in Chapter 4.5.

Finally, we mask-out pixels with a static appearance between consecutive frames, which includes scenes with no relative motion. In doing so the network ignores pixels which move at the same velocity as the camera and also frames in which the camera does not move. According to [80], this is accomplished by removing those pixels which have an unwarped photometric loss smaller than the corresponding warped photometric loss.

Semantic term. The standard cross-entropy loss between the predicted and proxy pixel-wise semantic labels is used as semantic term:

$$\mathcal{L}_{sem}(p) = -\sum_{i=1}^{N} \hat{S}^i(p) \log(S_t^i(p)) \tag{C.2}$$

where N is the number of semantic classes, S_t^i is the score predicted by DSNet and $\hat{S}^i$ the ground-truth proxy label for the class i of the pixel p in the target frame t. Moreover, as proposed in [352], we employ a cross-task loss to tighten the link between the tasks of depth and semantic segmentation:

$$\mathcal{L}_{cdd}(p) = sgn(|\delta_x S_t(p)|) \cdot e^{-|\frac{\delta_x D_t(p)}{D_t(p)}|} + sgn(|\delta_y S_t(p)|) \cdot e^{-|\frac{\delta_y D_t(p)}{D_t(p)}|} \tag{C.3}$$

where sgn is the sign operator, D_t is the predicted depth map by DSNet, S_t the predicted semantic class for the pixel p and δ_x and δ_y are the horizontal and vertical gradients.

Hence, the total loss used to train DSNet is a weighted combination of the above losses:

$$\mathcal{L} = \lambda_1 \mathcal{L}_{ap}^{D} + \lambda_2 \mathcal{L}_{smooth} + \lambda_3 \mathcal{L}_{sem} + \lambda_4 \mathcal{L}_{cdd} \tag{C.4}$$

where λ_1, λ_2, λ_3 and λ_4 are hyper-parameters. In our experiments, we set $\lambda_1 = 1$, $\lambda_2 = 0.1$, $\lambda_3 = 1$ and $\lambda_4 = 0.1$.

Optical Flow term. We train a the first instance of the optical flow network (OFNet) using the same photometric loss as for DSNet:

$$\mathcal{L}_{ap}^{OF} = \sum_{p} \alpha \mathcal{L}_{DSSIM} + (1-\alpha)|I_t - \tilde{I}_t^s| \tag{C.5}$$

In this case, however, $\tilde{I}_t^s$ is warped according to estimated flow. Akin to DSNet, we set $\alpha = 0.85$.

Self-Distilled Optical Flow term. The self-distilled optical flow network (SD-OFNet), instead, is trained in a quite different manner. In fact, given the optical flow $F_{t\to s}$ predicted by OFNet, the rigid flow $F_{t\to s}^{rigid}$ and the mask M defined in Equation 6.7, we leverage the optical flow in the regions where $F_{t\to s}$ and $F_{t\to s}^{rigid}$ are similar as well as moving objects, while we rely on the rigid flow for the remaining areas (e.g., occlusions due to camera motion). We can distinguish the former regions from the latter ones looking at M. Moreover, we also apply a photometric term ψ on the predicted optical flow $SF_{t\rightarrow s}$. We use a $L1$ loss as ϕ. The final loss $\mathcal{L}$ to train SD-OFNet is described by Equation 6.8.

During training, $F_{t\rightarrow s}$, $F_{t\rightarrow s}^{rigid}$, M and the input images of SD-OFNet are randomly cropped to 416×128 before computing $\mathcal{L}$: in doing so, the errors in occluded areas of $F_{t\rightarrow s}$ due to camera motions, are less to appear and impact the training process. Finally, to ameliorate the photometric loss term, the image $\tilde{I}_t^{SF}$ is obtained by padding $SF_{t\rightarrow s}$ at first, which is predicted at 416×128, to original resolution (e.g., 640×192), then using this flow to warp the full resolution I_s at I_t coordinates and finally extracting from this image the same crop as used before. This simple strategy allows to leverage a complete image, since otherwise the cropped image would suffer from motion occlusions near boundaries. Moreover, we highlight that SD-OFNet is initialized to the OFNet weights, i.e. those found during the above described OFNet training based on the standard photometric loss. When training SD-OFNet, only its weights are updated, and OFNet is kept frozen.

C.3 Training protocol

Similarly to [347], we employ a two stage learning process to facilitate the network optimisation process. At first, we train DSNet and CamNet simultaneously, then we train OFNet by the self-distillation paradigm described in 6.2.3. For both stages, we use a batch size of 4 and resize input images to 640×192 for KITTI and to 768×384 for Cityscapes pre-training, optimizing the output of the networks at the highest resolution only. Nonetheless, Chapter 6 contains also additional experimental results of different input resolutions where specified. We use the Adam optimizer [139] with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. Finally, we set both τ and ξ, defined in Equations 6.9 and 6.6 respectively, to be 0.5.

Depth, Pose, Intrinsics and Semantic Segmentation. In order to train DSNet and CamNet we employ sequences of 3 consecutive frames and semantic proxy labels yielded by a state-of-the art architecture [48] trained on Cityscapes with ground-truth labels. We trained DSNet and CamNet for 300K iterations, setting the initial learning rate to 10^{-4}, manually halved after 200K, 250K and 275K steps. We apply data augmentation to images as in [79]. Training takes $\sim$ 20 hours on a Titan Xp GPU.

Optical Flow. We train OFNet with the following procedure. We perform 200K training steps with an initial learning rate of 10^{-4}, halved every 50K until convergence. Moreover, we apply strong data augmentation consisting in random horizontal and vertical flip, crops, random time order switch and, peculiarly, time stop, replacing all I_s with I_t to learn a zero motion vector. This configuration requires about 13 hours on a Titan Xp GPU with the standard 640×192 resolution. Once obtained a competitive network in non-occluded regions we train a more robust optical flow network, denoted as SD-OFNet, starting from pre-learned weights and the same structure of OFNet by distilling knowledge from OFNet and rigid flow computed by DSNet using the total mask M and 416×128 random crops applied to $F_{t\rightarrow s}$, $F_{t\rightarrow s}^{rigid}$, M and RGB images. We train SD-OFNet for $15K$ steps only with a learning rate of 2.5×10^{-5} halved after 5K, 7.5K, 10K and 12.5K steps, setting α_r to 0.025 and α_d to 0.2. At test-time, we rely on SD-OFNet only.

Appendix D

Additional details for Chapter 11

D.1 Implementation details

Our framework is implemented in PyTorch and it is trained using a single NVIDIA 3090 GPU. All modules are initialized from scratch. We use Adam [139] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We adopted the popular VGG13 [276] as feature extractor for both the RGB input image $\mathcal{I}_l$ as well as the correspondent noisy disparity map D. Notice that the two feature extractors do not share weights. The MLPs in charge of estimating the final disparity map, $\widetilde{D}$, are implemented similarly to [258], though we replace the ReLU activations of the hidden layers by Sine functions [277].

Balanced Setup. In the balanced stereo setting, I_l and I_r are at the same spatial resolution, thus the stereo blackbox computes an initial disparity map D at that same image size. With reference to Figure 11.2, in such a scenario $D_\downarrow = D$ and $\mathcal{I}_l = \mathcal{I}_{l\downarrow}$, due to no downsampling/upsampling operations being needed. We trained our refinement architecture, counting approximately 22.9 million of parameters, on the SceneFlow dataset for 100 epochs setting the batch size to 8, using random crops of 384×384 size as input to the network and sampling randomly and uniformly $N = 30,000$ continuous 2D locations from each crop. We employed a learning rate of 10^{-4}, halved after 80 epochs. During training, we randomly alternate as stereo blackbox two popular traditional stereo algorithms, namely SGM [101] and AD-Census [351]. Moreover, we also provided a input corrupted ground-truth disparities obtained by adding different randomly-generated nuisances, such as Gaussian noise. By doing so, we force the network to handle a variety of different noisy patterns. We set the maximum disparity value to 256. A strong data augmentation procedure is applied to the RGB image as well as to the input disparity map, both normalized in the $[0, 1]$ interval. In particular, the input disparities -and, accordingly, the ground-truth maps - are scaled by a factor randomly drawn from $[0.2, 3]$. The training process takes approximately 24 hours.

Unbalanced Setup. When $\kappa \neq 1$ we tackle the unbalaced setting, where the two images of a stereo pair are captured at different image resolutions. In particular, without loss of generality, we assume $\kappa > 1$, that is $\mathcal{I}_l$ having a resolution higher than $\mathcal{I}_r$. We experiment with stereo black-boxes realized by existing methods, both traditional as well as based on deep networks. As such methods process two images at the same size in order to produce a disparity map, we first downsample the reference image $\mathcal{I}_l$ to match the lower resolution image $\mathcal{I}_r$ using bilinear interpolation, then naïvely upsample the computed disparity map $D_\downarrow$ by nearest neighbor interpolation so as to match the resolution of $\mathcal{I}_l$ and obtain the actual map, D, fed as input to our network. In our experiments, we use the UnrealStereo4K dataset for training following the protocol already described for the balanced setup but for the number of epochs and the crop size, set to 200 and 768×768, respectively.

As our architecture is not specifically designed to process very high resolution images, such as those provided by the UnrealStereo4K dataset (8 Mpx), in order to train and validate our model we resize the reference image, $\mathcal{I}_l$, to 1920×1080 resolution. Thus, in the case of UnrealStereo4K, we receive a full-res $\mathcal{I}_l$ image and a low-res $\mathcal{I}_r$ (according to a certain unbalance factor κ) and then feed $\phi_{\mathcal{I}}$ with a half-res $\mathcal{I}_l$, thereby constraining also the resolution of D to be 1920×1080. Nonetheless, to assess the ability of our architecture to handle very high resolution stereo pairs, at test time we evaluate the performance by comparing the predicted disparities to the ground-truth available for the original full-res (8 Mpx) stereo pair. In fact, seamlessly evaluating at whatever output resolution regardless of the size(s) of the input images is a key trait of our architecture enabled by the proposed continuous formulation. In the experiments, we will compare with PSMNet [43] and HSMNet [337], trained respectively for 60 and 200 epochs with batches of 4 and 12 samples and the same crop size used for our model.

D.2 Qualitative results

We now present additional qualitative results obtained using our neural refinement network on different stereo datasets in the balanced setting. Figure D.1 shows a qualitative comparison between our model and GANet [354] on *PianoL* stereo pair from Middlebury v3, when both trained on the SceneFlow dataset. We can appreciate our our refinement network produces fewer errors and achieves an lower bad2 overall score, thus better generalizing to real images.

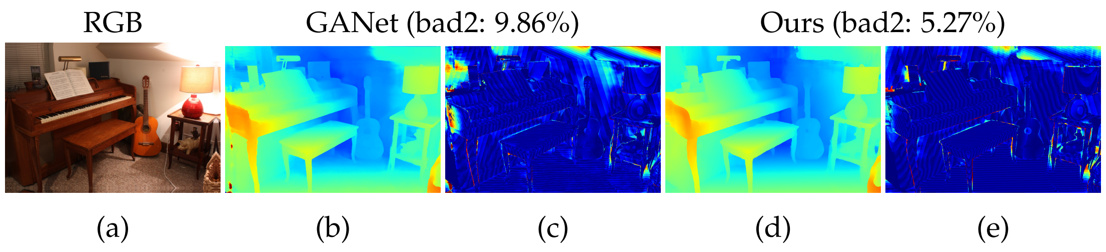


FIGURE D.1: **Qualitative comparison with GANet [354].** From left to right: reference image (a), disparity and error maps by GANet (b,c) and our method (d,e).

Figure D.2 collects some examples from the SceneFlow testing split, showing from left to right the reference image, the raw disparity map estimated by SGM and the final output by our network. Figures D.3, D.4 and D.5 report additional qualitative results on real datasets, respectively KITTI 2015, Middlebury 2014 and ETH3D, highlighting once again the outstanding zero-shot generalization performance achieved by our network, trained on synthetic datasets only.

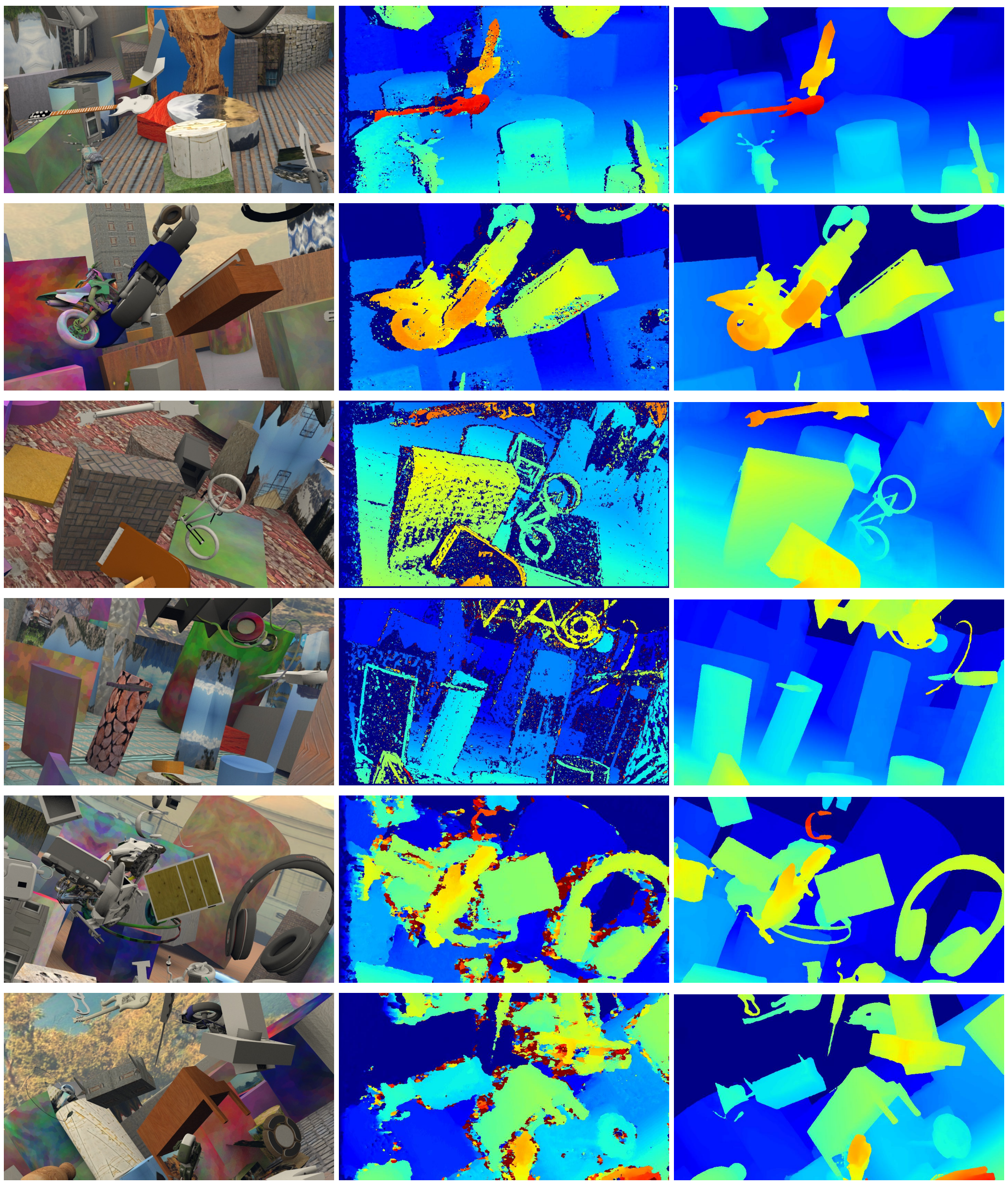


FIGURE D.2: **Qualitative results on the SceneFlow test set.** We report qualitative results of our neural disparity refinement network on the SceneFlow testing set. From left to right, the RGB input image, the noisy input disparity map computed by SGM [101] (rows 1-2), AD-Census [351] (rows 3-4), C-CNN [179] (rows 5-6) and the corresponding refined disparity estimated by our network.

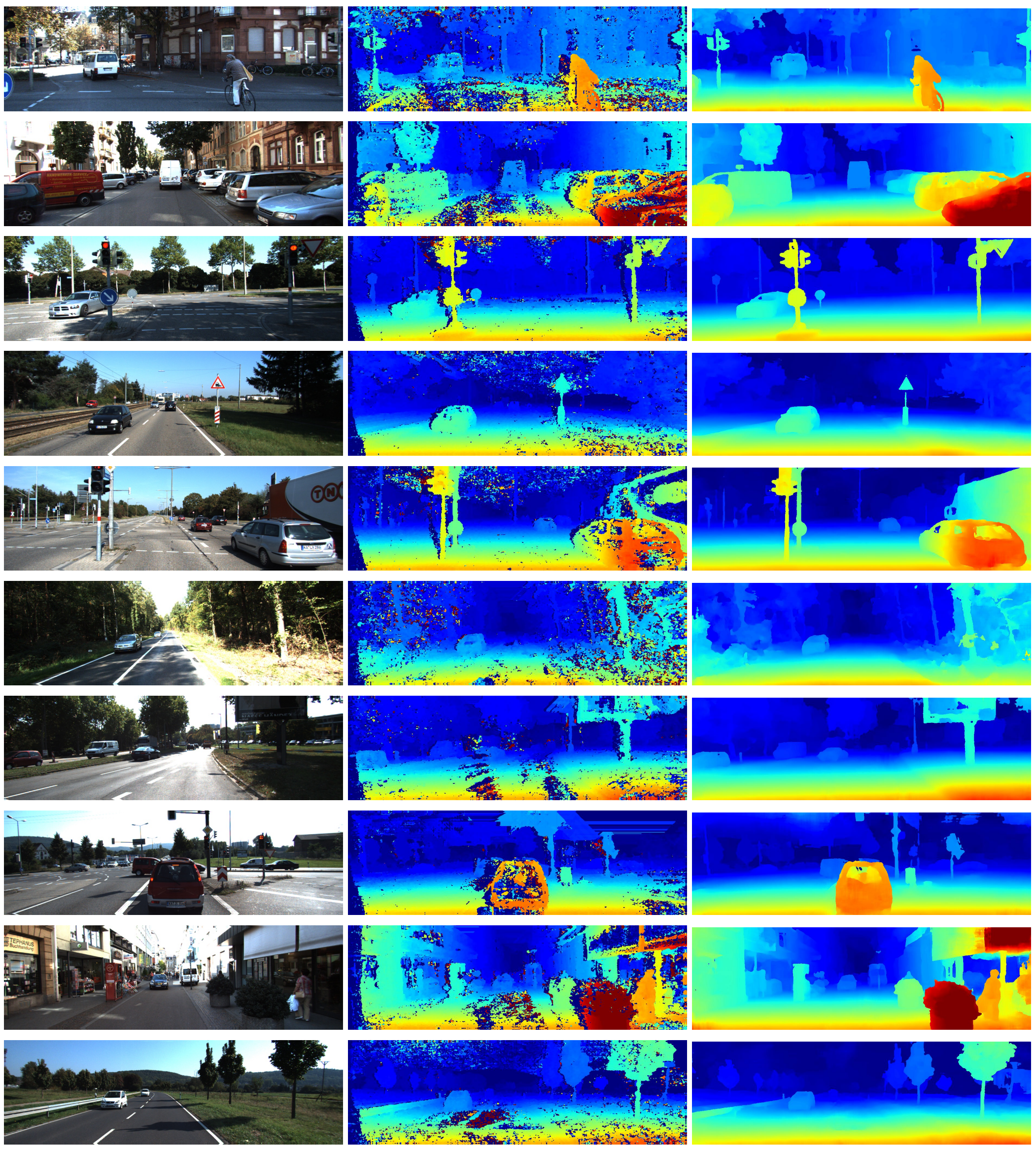


FIGURE D.3: **Qualitative results on the KITTI 2015 training set**. Here, we show qualitative results concerning the generalization capability of our network (pre-trained on SceneFlow) on the KITTI 2015 training set. From left to right, the RGB input image, the noisy input disparity map computed by SGM [101] and the refined disparity estimated by our network.

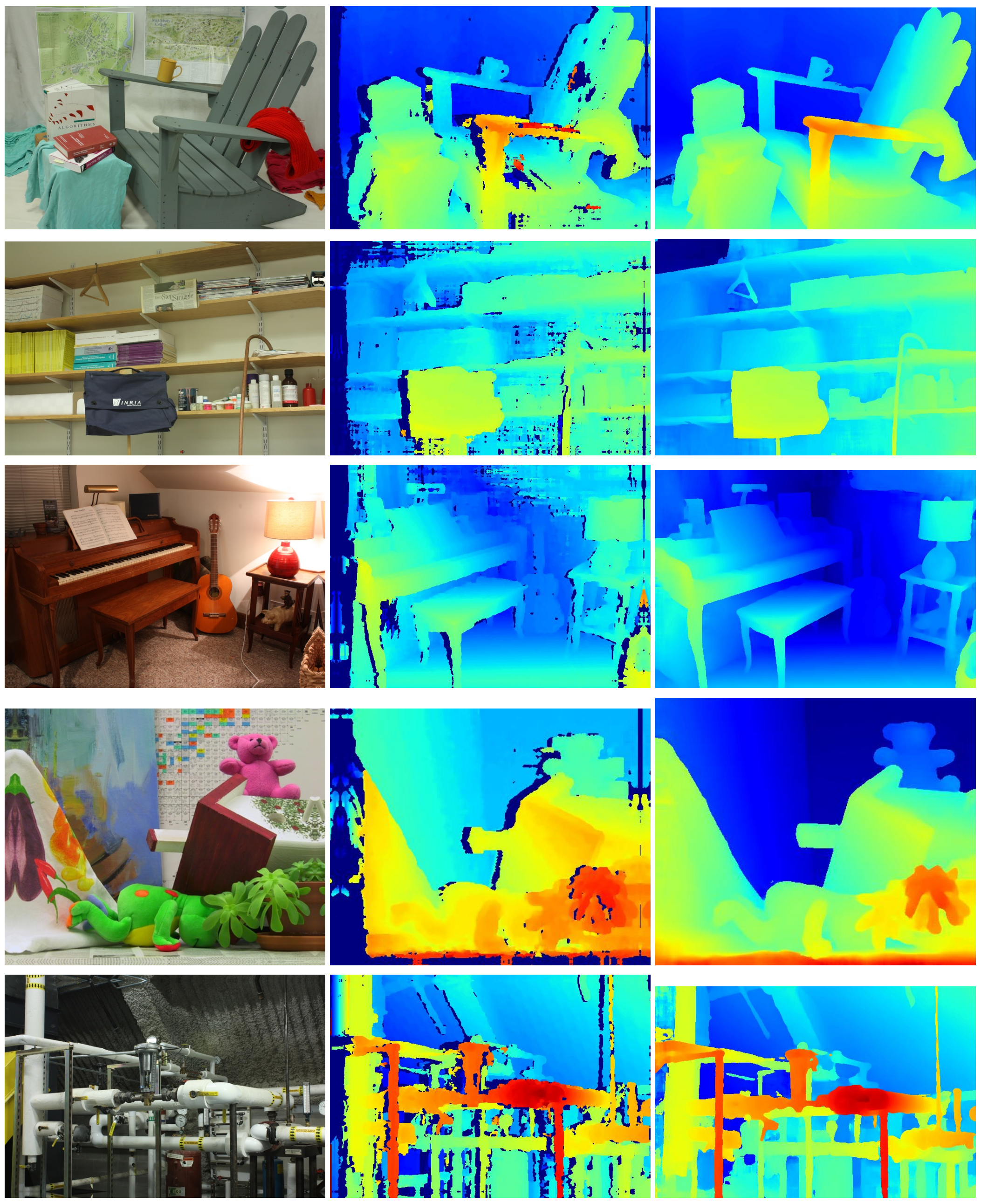


FIGURE D.4: **Qualitative results on the Middlebury v3 training Set**. Here, we show qualitative results concerning the generalization capability of our network (pre-trained on SceneFlow) on the Middlebury v3 training set. From left to right, the RGB input image, the noisy input disparity map computed by SGM [101] and the refined disparity estimated by our network.

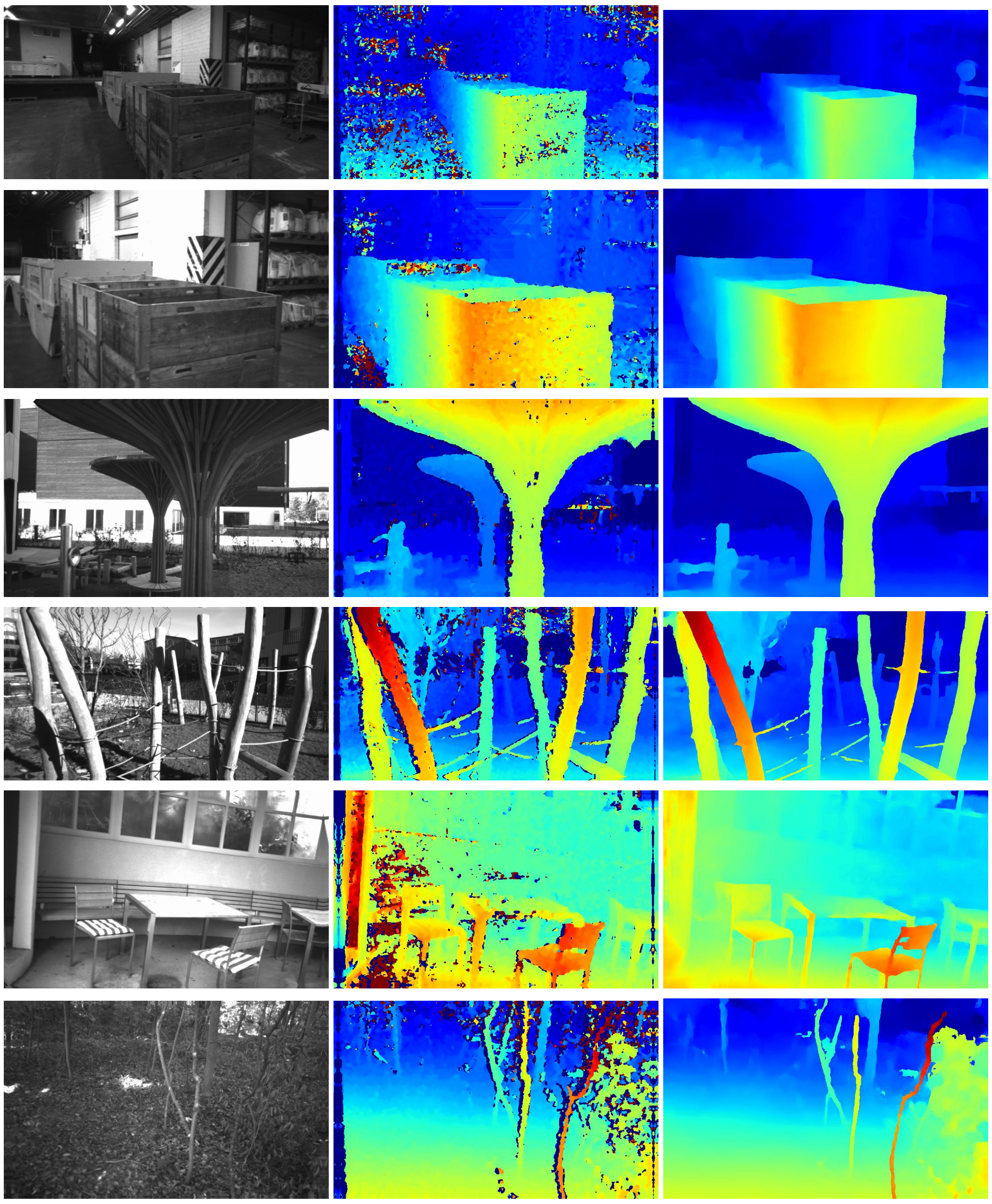


FIGURE D.5: **Qualitative results on the ETH3D training set**. Here, we show qualitative results concerning the generalization capability of our network (pre-trained on Scene-Flow) on the ETH3D training set. From left to right, the RGB input image, the noisy input disparity map computed by SGM [101] and the refined disparity estimated by our network.

D.3 Calibration and rectification of an unbalanced stereo rig

In Chapter 11 we have introduced the concept of unbalanced stereo, i.e. when the images of the pair are captured by heterogeneous sensors. To mimic this setup, we employed synthetic data and image transformations (e.g. resize).

Now, we describe in detail how to accurately calibrate an unbalanced stereo rig, made of a high-resolution camera and one at lower-resolution collecting respectively frames $\mathcal{I}_l$ and $\mathcal{I}_r$. Such a rig is characterized by an unbalance factor κ, defined as the ratio between $\mathcal{I}_l$ and $\mathcal{I}_r$ widths. The calibration process allows us to *rectify* the frame pairs acquired by the rig. In such a setting, the rectification constraint shall be understood to hold up to a scale factor, i.e. $\mathcal{I}_l$ and $\mathcal{I}_r$ turn out to be rectified whenever resized to the same - arbitrary - shape.

We first calibrate each camera separately using the pinhole camera model. The well-known distortion-free projective transformation performed by a pinhole camera model is given by:

$$p = A\mathcal{R}\mathcal{T}P_w \tag{D.1}$$

where P_w is a 3D point expressed w.r.t. the world reference frame (WRF), p is a 2D pixel in the image plane, A is the intrinsic parameters matrix and $\mathcal{R}$, $\mathcal{T}$ are the rotation and translation from the world reference frame (WRF) to the camera reference frame (CRF), respectively.

However, real lenses have radial and tangential distortions. We follow the lens distortion model adopted in the OpenCV library, where such a distortion is modelled through a vector of parameters $Dist = k_1, k_2, k_3, p_1, p_2$, with k_1, k_2, k_3 denoting the radial distortion parameters and p_1, p_2 the tangential distortion parameters respectively.

Given a known pattern (e.g., a chessboard), we can find in the images a set of key-points (e.g., inner corners of the chessboard) for which we know the exact 3D position in the WRF and, accordingly, build a set of 2D-3D correspondences which allows for inferring camera parameters through calibration. We estimate the 2D coordinates of the corners, namely p_L, p_R, in images acquired by L, R cameras, respectively, by using a standard corner detection algorithm. By calibrating each camera of the rig independently, we estimate their intrinsic matrices A_L, A_R and the lens distortion parameters $Dist_L, Dist_R$ of the L and R cameras, respectively. Given the intrinsic and distortion parameters, we can undistort the images to perform a stereo calibration of the stereo rig. We can thus estimate the rotations $\mathcal{R}_{LR}$ and translations $\mathcal{T}_{LR}$, from the L to R CRFs.

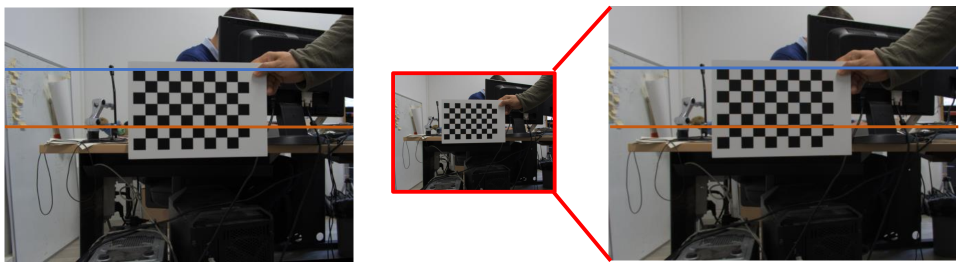


FIGURE D.6: **Unbalanced rectified L and R images.** Upsampling only the low-resolution R image to match the resolution of L yields a rectified stereo pair at the highest resolution.

Typically when estimating a stereo rectification transformation, we assume to have both cameras at the same resolution and a similar Field of View (FOV). The new projection matrix is typically found as the mean between the initial intrinsic matrixes of the two cameras. However, as in our case, the R camera has dramatically different characteristics compared to other cameras, directly performing the traditional rectification process would yield poor results. Indeed, we would need to perform harsh downsampling of the L image or a large upsampling of the R image to get rectified images. In contrast, we would like our rectified images to preserve their original resolution with the smallest amount of interpolation. Thus, we define the concept of *unbalanced rectification*, which allows for obtaining rectified images by performing only up-sampling or down-sampling operations, as illustrated in Figure D.6. To achieve the best possible rectification with a small amount of interpolation, we use the following procedure. First, we calculate the Horizontal Field Of Views $HFOV_L$ and $HFOV_R$ using the focal length f_L, f_R (known from the intrinsic matrices) of the L and R cameras, respectively. Then, we find the camera with the smaller $HFOV$, which will define an upper bound of the common visible area between the two cameras. We denote the camera with the smaller $HFOV$ as j while the other one as i.

$$\begin{cases} i = L, j = R & \text{if } HFOV_R < HFOV_L \\ i = R, j = L & \text{if } HFOV_L < HFOV_R \end{cases} \tag{D.2}$$

Then, we modify the intrinsic parameters of i to simulate a crop and scale of its images so as to match the $HFOV$, Aspect Ratio (AR) and size of j, and eventually calculate the rectification transformation with these parameters.

Hence, we calculate the new width and height of i, $\hat{W}_i$ and $\hat{H}_i$, which we use to crop the image with the larger $HFOV$ to match the smaller $HFOV$ one and to preserve the aspect ratio as follows:

$$\hat{W}_i = 2 \tan \frac{HFOV_j}{2} f_i \tag{D.3}$$

$$\hat{H}_i = \frac{H_j}{W_j} \hat{W}_i \tag{D.4}$$

Then, we modify the intrinsic parameters of i to simulate the crop and resize to match the resolution of j as follows:

$$\hat{A}_i = \begin{bmatrix} f_x^i \cdot \frac{W_j}{\hat{W}_i} & 0 & (u_0^i - \frac{W_i - \hat{W}_i}{2}) \cdot \frac{W_j}{\hat{W}_i} \\ 0 & f_y^i \cdot \frac{H_j}{\hat{H}_i} & (v_0^i - \frac{H_i - \hat{H}_i}{2}) \cdot \frac{H_j}{\hat{H}_i} \\ 0 & 0 & 1 \end{bmatrix}$$

We estimate the rectification transformation as we would have two cameras of height H_j and width W_j, finding the new intrinsic $A_{L_{rect}}$ and $A_{R_{rect}}$, and the rotations $\hat{\mathcal{R}}_{Lrect}, \hat{\mathcal{R}}_{Rrect}$, of L and R to map the initial image plane into the rectified image plane. Finally, as we have estimated the intrinsic matrixes at the resolution of j , we rescale $A_{i_{rect}}$ (i.e., focal and piercing point) with a vertical and horizontal scale factors equal to $\frac{\hat{H}_i}{H_j}$ and $\frac{\hat{W}_i}{W_j}$, respectively.

Figure D.7 shows an example of raw, unbalanced stereo pair on the left, with $\mathcal{I}_l$ and $\mathcal{I}_r$ acquired respectively by two cameras at very different resolutions. The calibration procedure described in the reminder allows to rectify them, as shown on the right.

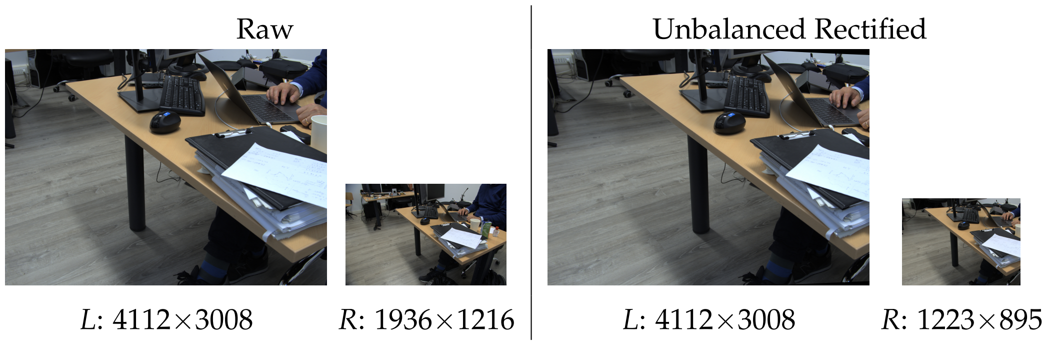


FIGURE D.7: **Example of unbalanced stereo pair.** On left, raw images acquired by two very different cameras, respectively at 412 × 3008 and 1936 × 1216 resolution. On right, images rectified according to unbalanced calibration and rectification.

This allows us to design a custom unbalanced stereo rig, emulating the setup commonly available on mobile smartphones, that we use to collect additional samples over which we can qualitatively appreciate the effectiveness of our Neural Disparity Refinement framework. Figure D.8 shows four examples acquired in an indoor environment, framing from left to right the high-resolution image used as reference, the initial disparity map computed by means of SGM and the outcome of our network.

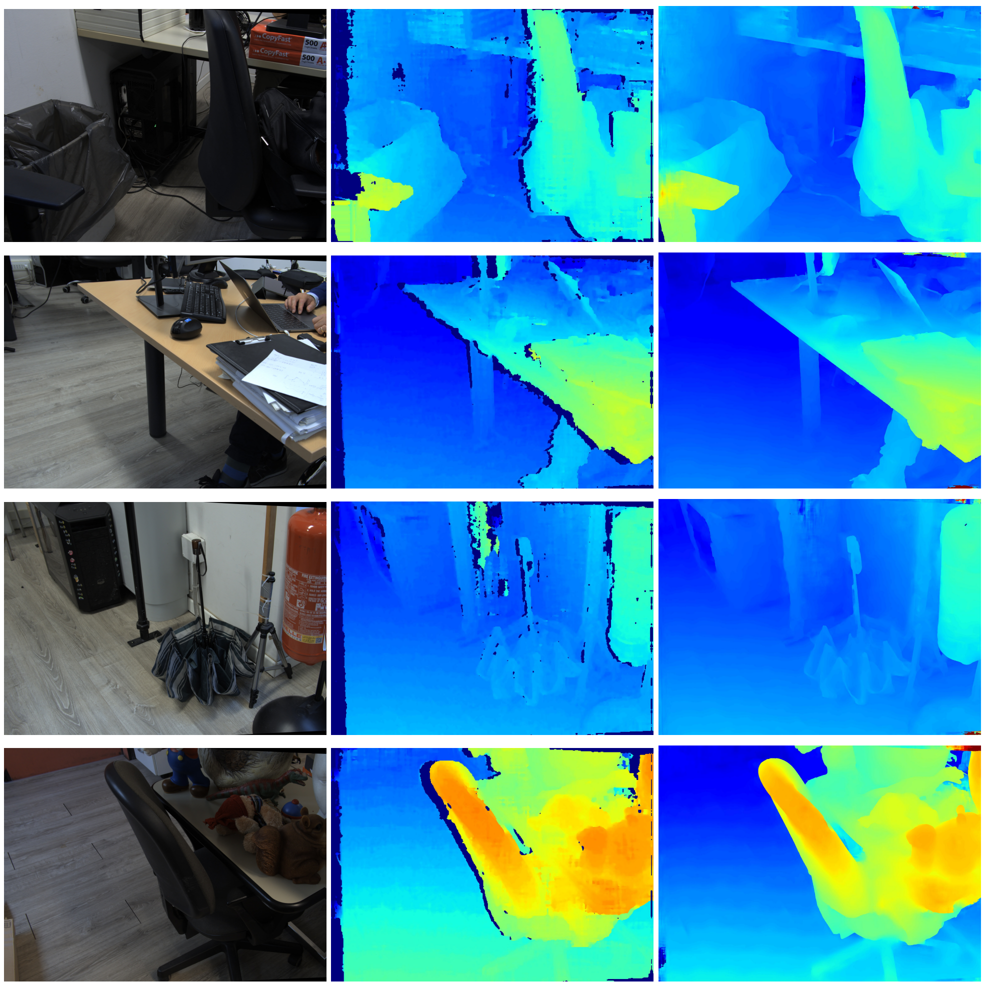


FIGURE D.8: **Qualitative results on a real unbalanced stereo setup**. We show qualitative results obtained by our network (trained on the synthetic SceneFlow dataset only) on real-world images captured using an unbalanced stereo setting featuring two cameras at 4112 × 3008 and 1223 × 895 resolution. From left to right, we show the high-res RGB image, the initial disparity map computed by SGM [101] and the estimated disparity at 4112 × 3008 resolution.

Bibliography

[1] A.N.Prasad, Kabir Mamun, F. Islam, and H. Haqva. “Smart Water Quality Monitoring System”. In: 2015.

[2] Ahmed Abdelgawad and Kumar Yelamarthi. “Structural health monitoring: Internet of things application”. In: *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*. 2016.

[3] Donato Abruzzese, Andrea Micheletti, Alessandro Tiero, Manuel Cosentino, Damiano Forconi, Gianmarco Grizzi, Gianluca Scarano, Sreymom Vuth, and Pierluigi Abiuso. “IoT sensors for modern structural health monitoring. A new frontier”. In: *Procedia Structural Integrity* 25 (2020), pp. 378–385.

[4] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. “SLIC superpixels compared to state-of-the-art superpixel methods”. In: *Transactions on Pattern Analysis and Machine Intelligence* 34.11 (2012), pp. 2274–2282.

[5] Maya Aghaei, Matteo Bustreo, Yiming Wang, Gianluca Bailo, Pietro Morerio, and Alessio Del Bue. “Single Image Human Proxemics Estimation for Visual Social Distancing”. In: *Winter Conference on Applications of Computer Vision*. 2021.

[6] Afiq Harith Ahamad, Norliza Zaini, and Mohd Fuad Abdul Latip. “Person Detection for Social Distancing and Safety Violation Alert based on Segmented ROI”. In: *International Conference on Control System, Computing and Engineering*. IEEE. 2020.

[7] Filippo Aleotti, Matteo Poggi, and Stefano Mattoccia. “Learning optical flow from still images”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2021.

[8] Filippo Aleotti, Fabio Tosi, Matteo Poggi, and Stefano Mattoccia. “Generative Adversarial Networks for unsupervised monocular depth prediction”. In: *European Conference on Computer Vision Workshops*. Springer. 2018.

[9] Filippo Aleotti, Fabio Tosi, Pierluigi Zama Ramirez, Matteo Poggi, Samuele Salti, Luigi Di Stefano, and Stefano Mattoccia. “Distilled Semantics for Comprehensive Scene Understanding from Videos”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2020.

[10] Filippo Aleotti, Fabio Tosi, Pierluigi Zama Ramirez, Matteo Poggi, Samuele Salti, Luigi Di Stefano, and Stefano Mattoccia. “Neural Disparity Refinement for Arbitrary Resolution Stereo”. In: *International Conference on 3D Vision*. 2021.

[11] Filippo Aleotti, Fabio Tosi, Li Zhang, Matteo Poggi, and Stefano Mattoccia. “Reversing the cycle: self-supervised deep stereo through enhanced monocular distillation”. In: *European Conference on Computer Vision*. Springer. 2020.

[12] Filippo Aleotti, Giulio Zaccaroni, Luca Bartolomei, Matteo Poggi, Fabio Tosi, and Stefano Mattoccia. “Real-Time Single Image Depth Perception in the Wild with Handheld Devices”. In: *Sensors* 21.1 (2021).

[13] Padmanabhan Anandan. “A computational framework and an algorithm for the measurement of visual motion”. In: *International Journal of Computer Vision* 2.3 (1989), pp. 283–310.

[14] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodik, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. “Real-time video analytics: The killer app for edge computing”. In: *computer* 50.10 (2017), pp. 58–67.

[15] Ranjan Anurag, Varun Jampani, Kihwan Kim, Deqing Sun, Jonas Wulff, and Michael J. Black. “Competitive collaboration: Joint unsupervised learning of depth, camera motion, optical flow and motion segmentation”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2019.

[16] *Apple ARkit*. URL: `developer.apple.com/augmented-reality/`.

[17] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. “Segnet: A deep convolutional encoder-decoder architecture for image segmentation”. In: *Transactions on Pattern Analysis and Machine Intelligence* 39.12 (2017), pp. 2481–2495.

[18] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. “A database and evaluation methodology for optical flow”. In: *International Journal of Computer Vision* 92.1 (2011), pp. 1–31.

[19] Christian Banz, Sebastian Hesselbarth, Holger Flatt, Holger Blume, and Peter Pirsch. “Real-time stereo vision system using semi-global matching disparity estimation: Architecture and FPGA-implementation”. In: *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*. 2010, pp. 93–101.

[20] John Barron, David Fleet, and S. Beauchemin. “Performance Of Optical Flow Techniques”. In: *International Journal of Computer Vision* 12 (Feb. 1994), pp. 43–77.

[21] Antoine Basset, Patrick Bouthemy, and Charles Kervrann. “Recovery of motion patterns and dominant paths in videos of crowded scenes”. In: 2014.

[22] Konstantinos Batsos and Philipos Mordohai. “RecResNet: A Recurrent Residual CNN Architecture for Disparity Map Enhancement”. In: *International Conference on 3D Vision*. 2018.

[23] Peter N Belhumeur. “A Bayesian approach to binocular steropsis”. In: *International Journal of Computer Vision* 19.3 (1996), pp. 237–260.

[24] Shariq Farooq Bhat, Ibraheem Alhashim, and Peter Wonka. “Adabins: Depth estimation using adaptive bins”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2021.

[25] Jia-Wang Bian, Zhichao Li, Naiyan Wang, Huangying Zhan, Chunhua Shen, Ming-Ming Cheng, and Ian Reid. “Unsupervised Scale-consistent Depth and Ego-motion Learning from Monocular Video”. In: *Conference on Neural Information Processing Systems*. 2019.

[26] Sizhen Bian, Bo Zhou, Hymalai Bello, and Paul Lukowicz. “A wearable magnetic field based proximity sensing system for monitoring COVID-19 social distancing”. In: *International Symposium on Wearable Computers*. 2020, pp. 22–26.

[27] Stan Birchfield and Carlo Tomasi. “A pixel dissimilarity measure that is insensitive to image sampling”. In: *Transactions on Pattern Analysis and Machine Intelligence* 20.4 (1998), pp. 401–406.

[28] Stan Birchfield and Carlo Tomasi. “Depth discontinuities by pixel-to-pixel stereo”. In: *International Journal of Computer Vision* 35.3 (1999), pp. 269–293.

[29] Michael J Black and Padmanabhan Anandan. “A framework for the robust estimation of optical flow”. In: *International Conference on Computer Vision*. IEEE. 1993.

[30] John Boardman and Robert Evans. “The measurement, estimation and monitoring of soil erosion by runoff at the field scale: Challenges and possibilities with particular reference to Britain”. In: *Progress in Physical Geography: Earth and Environment* (2020).

[31] Yuri Boykov and Vladimir Kolmogorov. “An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision”. In: *Transactions on Pattern Analysis and Machine Intelligence* 26.9 (2004), pp. 1124–1137.

[32] Yuri Boykov, Olga Veksler, and Ramin Zabih. “Fast approximate energy minimization via graph cuts”. In: *Transactions on Pattern Analysis and Machine Intelligence* 23.11 (2001), pp. 1222–1239.

[33] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[34] Thomas Brox, Christoph Bregler, and Jitendra Malik. "Large displacement optical flow". In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2009.

[35] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. "High accuracy optical flow estimation based on a theory for warping". In: *European Conference on Computer Vision*. Springer. 2004.

[36] Antoni Buades and Gabriele Facciolo. "Reliable Multiscale and Multiwindow Stereo Matching". In: *SIAM Journal on Imaging Sciences* (2015).

[37] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. "A naturalistic open source movie for optical flow evaluation". In: *European Conference on Computer Vision*. Springer, 2012.

[38] Changjiang Cai, Matteo Poggi, Stefano Mattoccia, and Philippos Mordohai. "Matching-space Stereo Networks for Cross-domain Generalization". In: *International Conference on 3D Vision*. 2020.

[39] Carlos Campos, Richard Elvira, Juan J. Gomez, José M. M. Montiel, and Juan D. Tardós. "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM". In: *arXiv preprint arXiv:2007.11898* (2020).

[40] Vincent Casser, Soeren Pirk, Reza Mahjourian, and Anelia Angelova. "Depth Prediction without the Sensors: Leveraging Structure for Unsupervised Learning from Monocular Videos". In: *AAAI Conference on Artificial Intelligence*. 2019.

[41] Vincent Casser, Soeren Pirk, Reza Mahjourian, and Anelia Angelova. "Unsupervised monocular depth and ego-motion learning with structure and semantics". In: *Conference on Computer Vision and Pattern Recognition Workshops*. 2019.

[42] Alexandros André Chaaraoui, Pau Climent-Pérez, and Francisco Flórez-Revuelta. "A review on vision techniques applied to human behaviour analysis for ambient-assisted living". In: *Expert Systems with Applications* 39.12 (2012), pp. 10873–10888.

[43] Jia-Ren Chang and Yong-Sheng Chen. "Pyramid Stereo Matching Network". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2018.

[44] Sirinthra Chantharaj, Kissada Pornratthanapong, Pitchayut Chitsinpchayakun, Teerapong Panboonyuen, Peerapon Vateekul, Siam Lawavirojwong, Panu Srestasathiern, and Kulsawasd Jitkajornwanich. "Semantic Segmentation On Medium-Resolution Satellite Images Using Deep Convolutional Networks With Remote Sensing Derived Indices". In: *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. 2018.

[45] Ping Chao, Chao-Yang Kao, Yu-Shan Ruan, Chien-Hsiang Huang, and Youn-Long Lin. "HarDNet: A Low Memory Traffic Network". In: *International Conference on Computer Vision*. IEEE, 2019.

[46] Chuangrong Chen, Xiaozhi Chen, and Hui Cheng. "On the Over-Smoothing Problem of CNN Based Disparity Estimation". In: *International Conference on Computer Vision*. IEEE, 2019.

[47] Liang Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. "Rethinking Atrous Convolution for Semantic Image Segmentation Liang-Chieh". In: *Transactions on Pattern Analysis and Machine Intelligence* 40 (4 2018). ISSN: 01628828.

[48] Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. "Searching for efficient multi-scale architectures for dense image prediction". In: *Conference on Neural Information Processing Systems*. 2018.

[49] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *Transactions on Pattern Analysis and Machine Intelligence* 40.4 (2017), pp. 834–848.

[50] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. “Semantic image segmentation with deep convolutional nets and fully connected crfs”. In: *arXiv preprint arXiv:1412.7062* (2014).

[51] Long Chen, Wen Tang, and Nigel John. “Self-Supervised Monocular Image Depth Learning and Confidence Estimation”. In: *arXiv preprint arXiv:1803.05530* (2018).

[52] Yuhua Chen, Cordelia Schmid, and Cristian Sminchisescu. “Self-supervised Learning with Geometric Constraints in Monocular Video: Connecting Flow, Depth, and Camera”. In: *International Conference on Computer Vision*. IEEE, 2019.

[53] Bowen Cheng, Maxwell D Collins, Yukun Zhu, Ting Liu, Thomas S Huang, Hartwig Adam, and Liang-Chieh Chen. “Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2020.

[54] Steven D Cochran and Gerard Medioni. “3-D surface description from binocular stereo”. In: *Transactions on Pattern Analysis and Machine Intelligence* 14.10 (1992), pp. 981–994.

[55] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. “The cityscapes dataset for semantic urban scene understanding”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2016.

[56] Marco Cristani, Alessio Del Bue, Vittorio Murino, Francesco Setti, and Alessandro Vinciarelli. “The visual social distancing problem”. In: *IEEE Access* 8 (2020), pp. 126876–126886.

[57] Ross Cutler and Matthew Turk. “View-based interpretation of real-time optical flow for gesture recognition”. In: *International Conference on Automatic Face and Gesture Recognition*. IEEE. 1998, pp. 416–421.

[58] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. “Imagenet: A large-scale hierarchical image database”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2009.

[59] Luigi Di Stefano, Massimiliano Marchionni, and Stefano Mattoccia. “A fast area-based stereo matching algorithm”. In: *Image and Vision Computing* 22.12 (2004), pp. 983–1005.

[60] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *International Conference on Learning Representations*. 2020.

[61] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. “Flownet: Learning optical flow with convolutional networks”. In: *International Conference on Computer Vision*. IEEE. 2015.

[62] Shivam Duggal, Shenlong Wang, Wei-Chiu Ma, Rui Hu, and Raquel Urtasun. “DeepPruner: Learning Efficient Stereo Matching via Differentiable PatchMatch”. In: *International Conference on Computer Vision*. 2019.

[63] Geoffrey Egnal and Richard P Wildes. “Detecting binocular half-occlusions: Empirical comparisons of five approaches”. In: *Transactions on Pattern Analysis and Machine Intelligence* 24.8 (2002), pp. 1127–1133.

[64] David Eigen and Rob Fergus. “Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture”. In: *International Conference on Computer Vision*. 2015.

[65] David Eigen, Christian Puhrsch, and Rob Fergus. “Depth map prediction from a single image using a multi-scale deep network”. In: *Advances in neural information processing systems*. 2014.

[66] Matteo Fabbri, Fabio Lanzi, Riccardo Gasparini, Simone Calderara, Lorenzo Baraldi, and Rita Cucchiara. "Inter-Homines: Distance-Based Risk Estimation for Human Safety". In: (2020).

[67] Rui Fan, Sicen Guo, Li Wang, and Mohammud Junaid Bocus. "Computer-Aided Road Inspection: Systems and Algorithms". In: *arXiv preprint arXiv:2203.02355* (2022).

[68] Roman Fedorov, Alessandro Camerada, Piero Fraternali, and Marco Tagliasacchi. "Estimating snow cover from publicly available images". In: *IEEE Transactions on Multimedia* (2016).

[69] Maxime Ferrera, Alexandre Boulch, and Julien Moras. "Fast Stereo Disparity Maps Refinement By Fusion of Data-Based And Model-Based Estimations". In: *International Conference on 3D Vision*. IEEE. 2019.

[70] Martin A Fischler and Robert C Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[71] Denis Fortun, Patrick Bouthemy, and Charles Kervrann. "Optical flow modeling and computation: A survey". In: *Computer Vision and Image Understanding* 134 (2015), pp. 1–21.

[72] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. "Deep ordinal regression network for monocular depth estimation". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2018.

[73] Yasutaka Furukawa and Carlos Hernández. "Multi-View Stereo: A Tutorial". In: *Foundations and Trends in Computer Graphics and Vision* 9.1-2 (2015), pp. 1–148.

[74] Abdalla Gad, Gasm ElBary, Mohammad Alkhedher, and Mohammed Ghazal. "Vision-based Approach for Automated Social Distance Violators Detection". In: *2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT)*. 2020, pp. 1–5.

[75] Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. "Unsupervised cnn for single view depth estimation: Geometry to the rescue". In: *European Conference on Computer Vision*. Springer. 2016.

[76] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. "Vision meets robotics: The KITTI dataset". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237.

[77] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? the kitti vision benchmark suite". In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2012.

[78] Spyros Gidaris and Nikos Komodakis. "Detect, replace, refine: Deep structured prediction for pixel wise labeling". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2017.

[79] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. "Unsupervised monocular depth estimation with left-right consistency". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2017.

[80] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J Brostow. "Digging into self-supervised monocular depth estimation". In: *International Conference on Computer Vision*. IEEE, 2019.

[81] *Google ARcore*. URL: developers.google.com/ar/discover.

[82] Ariel Gordon, Hanhan Li, Rico Jonschkowski, and Anelia Angelova. "Depth from Videos in the Wild: Unsupervised Monocular Depth Learning from Unknown Cameras". In: *International Conference on Computer Vision*. IEEE, 2019.

[83] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. "A survey of deep learning techniques for autonomous driving". In: *Journal of Field Robotics* (2020).

[84] Vitor Guizilini, Rares Ambrus, Sudeep Pillai, Allan Raventos, and Adrien Gaidon. "3D Packing for Self-Supervised Monocular Depth Estimation". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2020.

[85] Vitor Guizilini, Rui Hou, Jie Li, Rares Ambrus, and Adrien Gaidon. "Semantically-Guided Representation Learning for Self-Supervised Monocular Depth". In: *International Conference on Learning Representations*. 2020.

[86] Muhammad Shahzeb Khan Gul, Michel Bätz, and Joachim Keinert. "Pixel-Wise Confidences for Stereo Disparities Using Recurrent Neural Networks." In: *British Machine Vision Conference*. 2019, p. 23.

[87] Fatma Guney and Andreas Geiger. "Displets: Resolving Stereo Ambiguities Using Object Knowledge". In: *Conference on Computer Vision and Pattern Recognition*. 2015.

[88] Cengiz Güngör and Kenan Zengin. "A Survey On Augmented Reality Applications Using Deep Learning". In: *GE - International Journal Of Engineering Research* (2017).

[89] Xiaoyang Guo, Hongsheng Li, Shuai Yi, Jimmy Ren, and Xiaogang Wang. "Learning monocular depth by distilling cross-domain stereo networks". In: *European Conference on Computer Vision*. Springer. 2018.

[90] Xiaoyang Guo, Kai Yang, Wukui Yang, Xiaogang Wang, and Hongsheng Li. "Group-wise Correlation Stereo Network". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[91] Hela Hadhri, Flavien Vernier, Abdourrahmane M Atto, and Emmanuel Trouvé. "Time-lapse optical flow regularization for geophysical complex phenomena monitoring". In: *ISPRS Journal of photogrammetry and remote sensing* 150 (2019), pp. 135–156.

[92] Ralf Haeusler, Rahul Nair, and Daniel Kondermann. "Ensemble learning for confidence measures in stereo vision". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2013.

[93] David Hafner, Oliver Demetz, and Joachim Weickert. "Why is the census transform good for robust optic flow computation?" In: *International Conference on Scale Space and Variational Methods in Computer Vision*. Springer. 2013, pp. 210–221.

[94] Asaad Hakeem et al. "Video Analytics for Business Intelligence". In: *Video Analytics for Business Intelligence*. Ed. by Caifeng Shan, Fatih Porikli, Tao Xiang, and Shaogang Gong. Vol. 409. Studies in Computational Intelligence. Springer, 2012, pp. 309–354.

[95] Muhammed Sakib Hasan, Shahjalal Khandaker, Md. Shahid Iqbal, and Md. Monirul Kabir. "A Real-Time Smart Wastewater Monitoring System Using IoT: Perspective of Bangladesh". In: *2020 2nd International Conference on Sustainable Technologies for Industry 4.0 (STI)*. 2020.

[96] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN". In: *International Conference on Computer Vision*. IEEE, 2017.

[97] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask r-cnn". In: *International Conference on Computer Vision*. IEEE, 2017.

[98] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2016.

[99] Sheng He, Ruqin Zhou, Shenhong Li, San Jiang, and Wanshou Jiang. "Disparity Estimation of High-Resolution Remote Sensing Images with Dual-Scale Matching Network". In: *Remote Sensing* 13.24 (2021), p. 5050.

[100] Daniel Hernandez-Juarez, Alejandro Chacón, Antonio Espinosa, David Vázquez, Juan Carlos Moure, and Antonio M. López. "Embedded Real-time Stereo Estimation via Semi-Global Matching on the GPU". In: *International Conference on Computational Science*. 2016, pp. 143–153.

[101] Heiko Hirschmuller. "Stereo processing by semiglobal matching and mutual information". In: *Transactions on Pattern Analysis and Machine Intelligence* 30.2 (2008), pp. 328–341.

[102] Derek Hoiem, Alexei A Efros, and Martial Hebert. "Automatic photo pop-up". In: *ACM SIGGRAPH Computer Graphics*. 2005.

[103] Aleksander Holynski and Johannes Kopf. "Fast Depth Densification for Occlusion-aware Augmented Reality". In: *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*. Vol. 37. 6. ACM, 2018.

[104] Weixiang Hong, Qingpei Guo, Wei Zhang, Jingdong Chen, and Wei Chu. "LPSNet: A Lightweight Solution for Fast Panoptic Segmentation". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2021.

[105] Berthold KP Horn and Brian G Schunck. "Determining optical flow". In: *Techniques and Applications of Image Understanding*. Vol. 281. International Society for Optics and Photonics. 1981, pp. 319–331.

[106] *How Many Photos Will Be Taken in 2022?* https://blog.mylio.com/how-many-photos-taken-in-2022/.

[107] *How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read*. https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read.

[108] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[109] Xiaoyan Hu and Philippos Mordohai. "A quantitative evaluation of confidence measures for stereo vision". In: *Transactions on Pattern Analysis and Machine Intelligence* 34.11 (2012), pp. 2121–2133.

[110] Yinlin Hu, Yunsong Li, and Rui Song. "Robust Interpolation of Correspondences for Large Displacement Optical Flow". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2017.

[111] Yinlin Hu, Rui Song, and Yunsong Li. "Efficient Coarse-to-Fine PatchMatch for Large Displacement Optical Flow". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2016.

[112] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. "Snapshot ensembles: Train 1, get m for free". In: *International Conference on Learning Representations*. 2017.

[113] Yu Huang and Yue Chen. "Survey of State-of-Art Autonomous Driving Technologies with Deep Learning". In: *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. 2020.

[114] LI Hui and OU Jinping. "Structural Health Monitoring: From Sensing Technology Stepping to Health Diagnosis". In: *Procedia Engineering* (2011).

[115] Tak-Wai Hui and Chen Change Loy. "LiteFlowNet3: Resolving Correspondence Ambiguity for More Accurate Optical Flow Estimation". In: *European Conference on Computer Vision*. Springer. 2020.

[116] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. "A Lightweight Optical Flow CNN - Revisiting Data Fidelity and Regularization". In: *Transactions on Pattern Analysis and Machine Intelligence*. IEEE, 2020.

[117] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. "LiteFlowNet: A Lightweight Convolutional Neural Network for Optical Flow Estimation". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2018.

[118] Junhwa Hur and Stefan Roth. "Iterative Residual Refinement for Joint Optical Flow and Occlusion Estimation". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[119] Eddy Ilg, Ozgun Cicek, Silvio Galesso, Aaron Klein, Osama Makansi, Frank Hutter, and Thomas Brox. "Uncertainty estimates and multi-hypotheses networks for optical flow". In: *European Conference on Computer Vision*. Springer. 2018.

[120] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. "Flownet 2.0: Evolution of optical flow estimation with deep networks". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2017.

[121] Google Inc. *Use Raw Depth in your Android app*. 2008.

[122] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. "Spatial transformer networks". In: *Conference on Neural Information Processing Systems*. 2015.

[123] Mohammad R Jahanshahi, Farrokh Jazizadeh, Sami F Masri, and Burcin Becerik-Gerber. "Unsupervised approach for autonomous pavement-defect detection and quantification using an inexpensive depth sensor". In: *Journal of Computing in Civil Engineering* (2013).

[124] Ajay Jain, Ben Mildenhall, Jonathan T. Barron, Pieter Abbeel, and Ben Poole. "Zero-Shot Text-Guided Object Generation with Dream Fields". In: (2022).

[125] Joel Janai, Fatma Guney, Anurag Ranjan, Michael Black, and Andreas Geiger. "Unsupervised Learning of Multi-Frame Optical Flow with Occlusions". In: *European Conference on Computer Vision*. 2018.

[126] Huaizu Jiang, Deqing Sun, Varun Jampani, Zhaoyang Lv, Erik Learned-Miller, and Jan Kautz. "Sense: A shared encoder network for scene-flow estimation". In: *International Conference on Computer Vision*. IEEE, 2019.

[127] Zequn Jie, Pengfei Wang, Yonggen Ling, Bo Zhao, Yunchao Wei, Jiashi Feng, and Wei Liu. "Left-Right Comparative Recurrent Model for Stereo Matching". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2018.

[128] Joseph Johnson Jr, Shiblee Hasan, David Lee, Chris Hluchan, and Nazia Ahmed. "Social-distancing monitoring using portable electronic devices". In: *Technical Disclosure Commons* (2020).

[129] Adrian Johnston and Gustavo Carneiro. "Self-Supervised Monocular Trained Depth Estimation Using Self-Attention and Discrete Disparity Volume". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2020.

[130] Rico Jonschkowski, Austin Stone, Jon Barron, Ariel Gordon, Kurt Konolige, and Anelia Angelova. "What Matters in Unsupervised Optical Flow". In: *European Conference on Computer Vision* (2020).

[131] Sunghun Joung, Seungryong Kim, Kihong Park, and Kwanghoon Sohn. "Unsupervised stereo matching using confidential correspondence consistency". In: *Transactions on Pattern Analysis and Machine Intelligence* (2019).

[132] Sami Kaivonen and Edith C.-H. Ngai. "Real-time air pollution monitoring with sensors on city bus". In: *Digital Communications and Networks* (2020).

[133] Artúr István Károly, Péter Galambos, József Kuti, and Imre J. Rudas. "Deep Learning in Robotics: Survey on Model Structures and Training Strategies". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2021).

[134] Kevin Karsch, Ce Liu, and Sing Bing Kang. "Depth extraction from video using non-parametric sampling". In: *European Conference on Computer Vision*. Springer. 2012.

[135] Alex Kendall and Yarin Gal. "What uncertainties do we need in bayesian deep learning for computer vision?" In: *Conference on Neural Information Processing Systems*. Curran Associates Inc., 2017.

[136] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. "End-To-End Learning of Geometry and Context for Deep Stereo Regression". In: *International Conference on Computer Vision*. IEEE, 2017.

[137] Sameh Khamis, Sean Fanello, Christoph Rhemann, Adarsh Kowdle, Julien Valentin, and Shahram Izadi. "Stereonet: Guided hierarchical refinement for real-time edge-aware depth prediction". In: *European Conference on Computer Vision*. Springer, 2018.

[138] Sanghun Kim, Dong-gon Yoo, and Young Hwan Kim. "Stereo confidence metrics using the costs of surrounding pixels". In: *International Conference on Digital Signal Processing*. IEEE. 2014, pp. 98–103.

[139] D Kinga and J Ba Adam. "A method for stochastic optimization". In: *International Conference on Learning Representations*. 2015.

[140] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. "Panoptic feature pyramid networks". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[141] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. "Panoptic segmentation". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[142] Andreas Klaus, Mario Sormann, and Konrad Karner. "Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure". In: *International Conference on Pattern Recognition*. Vol. 3. IEEE. 2006, pp. 15–18.

[143] Maria Klodt and Andrea Vedaldi. "Supervising the new with the old: learning SFM from SFM". In: *European Conference on Computer Vision*. Springer, 2018.

[144] Yutaro Kobayashi, Yoshiaki Taniguchi, Youji Ochi, and Nobukazu Iguchi. "A System for Monitoring Social Distancing Using Microcomputer Modules on University Campuses". In: *International Conference on Consumer Electronics - Asia*. 2020, pp. 1–4.

[145] Claudia Kondermann, Rudolf Mester, and Christoph Garbe. "A statistical confidence measure for optical flows". In: *European Conference on Computer Vision*. Springer. 2008.

[146] Johannes Kopf, Xuejian Rong, and Jia-Bin Huang. "Robust Consistent Video Depth Estimation". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2021.

[147] Ivan Krasin et al. "OpenImages: A public dataset for large-scale multi-label and multi-class image classification." In: (2017).

[148] Satyam Kumar, Vikas Gautam, Amit Kumar, and Puja Kumari. "Social Distancing using Bluetooth Low Energy to Prevent the Spread of COVID-19". In: *2021 11th International Conference on Cloud Computing, Data Science Engineering (Confluence)*. 2021, pp. 563–567.

[149] Yevhen Kuznietsov, Marc Proesmans, and Luc Van Gool. "CoMoDA: Continuous Monocular Depth Adaptation Using Past Experiences". In: *Winter Conference on Applications of Computer Vision*. 2021.

[150] Hamid Laga, Laurent Valentin Jospin, Farid Boussaid, and Mohammed Bennamoun. "A survey on deep learning techniques for stereo-based depth estimation". In: *Transactions on Pattern Analysis and Machine Intelligence* (2020).

[151] Hsueh-Ying Lai, Yi-Hsuan Tsai, and Wei-Chen Chiu. "Bridging Stereo Matching and Optical Flow via Spatiotemporal Correspondence". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[152] Varsha Lakshmikantha, Anjitha Hiriyannagowda, Akshay Manjunath, Aruna Patted, Jagadeesh Basavaiah, and Audre Arlene Anthony. "IoT based smart water quality monitoring system". In: *Global Transitions Proceedings* (2021). International Conference on Computing System and its Applications (ICCSA- 2021).

[153] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. “Simple and scalable predictive uncertainty estimation using deep ensembles”. In: *Conference on Neural Information Processing Systems*. 2017.

[154] Georgios Lampropoulos, Euclid Keramopoulos, and Konstantinos Diamantaras. “Enhancing the functionality of augmented reality using deep learning, semantic web and knowledge graphs: A review”. In: *Visual Informatics* (2020).

[155] Juhyun Lee, Nikolay Chirkov, Ekaterina Ignasheva, Yury Pisarchyk, Mogan Shieh, Fabio Riccardi, Raman Sarokin, Andrei Kulik, and Matthias Grundmann. “On-device neural net inference with mobile gpus”. In: *arXiv preprint arXiv:1907.01989* (2019).

[156] Ang Li and Zejian Yuan. “Occlusion Aware Stereo Matching via Cooperative Unsupervised Learning”. In: *Asian Conference on Computer Vision*. Springer, 2018.

[157] Boying Li, Yuan Huang, Zeyu Liu, Danping Zou, and Wenxian Yu. “StructDepth: Leveraging the structural regularities for self-supervised indoor depth estimation”. In: *International Conference on Computer Vision*. IEEE, 2021.

[158] Gen Li and Joongkyu Kim. “DABNet: Depth-wise Asymmetric Bottleneck for Real-time Semantic Segmentation”. In: *British Machine Vision Conference*. BMVA, 2019.

[159] Hanhan Li, Ariel Gordon, Hang Zhao, Vincent Casser, and Anelia Angelova. “Unsupervised monocular depth learning in dynamic scenes”. In: *arXiv preprint arXiv:2010.16404* (2020).

[160] Shunkai Li, Xin Wu, Yingdian Cao, and Hongbin Zha. “Generalizing to the Open World: Deep Visual Odometry with Online Adaptation”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2021.

[161] Zhengqi Li, Tali Dekel, Forrester Cole, Richard Tucker, Noah Snavely, Ce Liu, and William T Freeman. “Learning the depths of moving people by watching frozen people”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[162] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. “Neural scene flow fields for space-time view synthesis of dynamic scenes”. In: *Conference on Computer Vision and Pattern Recognition*. 2021.

[163] Zhengqi Li and Noah Snavely. “MegaDepth: Learning Single-View Depth Prediction from Internet Photos”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2018.

[164] Justin Liang, Namdar Homayounfar, Wei-Chiu Ma, Yuwen Xiong, Rui Hu, and Raquel Urtasun. “Polytransform: Deep polygon transformer for instance segmentation”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2020.

[165] Zhengfa Liang, Yiliu Feng, Yulan Guo, Hengzhu Liu, Wei Chen, Linbo Qiao, Li Zhou, and Jianfeng Zhang. “Learning for Disparity Estimation Through Feature Constancy”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2018.

[166] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. “Microsoft coco: Common objects in context”. In: *European Conference on Computer Vision*. Springer. 2014.

[167] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. “Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[168] Haotian Liu, Rafael A Rivera Soto, Fanyi Xiao, and Yong Jae Lee. “Yolactedge: Real-time instance segmentation on the edge”. In: *International Conference on Robotics and Automation*. IEEE. 2021.

[169] Pengpeng Liu, Irwin King, Michael R Lyu, and Jia Xu. “Ddflow: Learning optical flow with unlabeled data distillation”. In: *AAAI Conference on Artificial Intelligence*. Vol. 33. 2019.

[170] Pengpeng Liu, Michael Lyu, Irwin King, and Jia Xu. “Selflow: Self-supervised learning of optical flow”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2019.

[171] Xiangbin Liu, Liping Song, Shuai Liu, and Yudong Zhang. “A review of deep-learning-based medical image segmentation methods”. In: *Sustainability* 13.3 (2021), p. 1224.

[172] Ziwen Liu, Rosie Brigham, Emily Rosemary Long, Lyn Wilson, Adam Frost, Scott Allan Orr, and Josep Grau-Bové. “Semantic segmentation and photogrammetry of crowdsourced images to monitor historic facades”. In: *Heritage Science* (2022).

[173] Arsal-Hanif Livoroi, Andrea Conti, Luca Foianesi, Fabio Tosi, Filippo Aleotti, Matteo Poggi, Flavia Tauro, Elena Toth, Salvatore Grimaldi, and Stefano Mattoccia. “On the Deployment of Out-of-the-Box Embedded Devices for Self-Powered River Surface Flow Velocity Monitoring at the Edge”. In: *Applied Sciences* 11.15 (2021).

[174] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2015.

[175] Ilya Loshchilov and Frank Hutter. “Decoupled weight decay regularization”. In: *arXiv preprint arXiv:1711.05101* (2017).

[176] Xiao Xin Lu. “A Review of Solutions for Perspective-n-Point Problem in Camera Pose Estimation”. In: *Journal of Physics: Conference Series* 1087 (2018).

[177] Bruce D Lucas, Takeo Kanade, et al. “An iterative image registration technique with an application to stereo vision”. In: Vancouver, British Columbia. 1981.

[178] Chenxu Luo, Zhenheng Yang, Peng Wang, Yang Wang, Wei Xu, Ram Nevatia, and Alan Yuille. “Every pixel counts++: Joint learning of geometry and motion with 3D holistic understanding”. In: *Transactions on Pattern Analysis and Machine Intelligence* (2019).

[179] Wenjie Luo, Alexander G Schwing, and Raquel Urtasun. “Efficient deep learning for stereo matching”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2016.

[180] Xuan Luo, Jia-Bin Huang, Richard Szeliski, Kevin Matzen, and Johannes Kopf. “Consistent Video Depth Estimation”. In: *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*. Vol. 39. 4. ACM, 2020.

[181] Yue Luo, Jimmy Ren, Mude Lin, Jiahao Pang, Wenxiu Sun, Hongsheng Li, and Liang Lin. “Single view stereo matching”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2018.

[182] Wei-Chiu Ma, Shenlong Wang, Rui Hu, Yuwen Xiong, and Raquel Urtasun. “Deep rigid instance scene flow”. In: *Conference on Computer Vision and Pattern Recognition*. 2019.

[183] Yuqing Ma, Xianglong Liu, Shihao Bai, Lei Wang, Dailan He, and Aishan Liu. “Coarse-to-Fine Image Inpainting via Region-wise Convolutions and Non-Local Correlation.” In: *International Joint Conferences on Artificial Intelligence*. 2019.

[184] Ziyang Ma, Kaiming He, Yichen Wei, Jian Sun, and Enhua Wu. “Constant time weighted median filtering for stereo matching and beyond”. In: *International Conference on Computer Vision*. IEEE, 2013.

[185] Oisin Mac Aodha, Ahmad Humayun, Marc Pollefeys, and Gabriel J Brostow. “Learning a confidence measure for optical flow”. In: *Transactions on Pattern Analysis and Machine Intelligence* 35.5 (2012), pp. 1107–1120.

[186] David JC MacKay. “A practical Bayesian framework for backpropagation networks”. In: *Neural computation* 4.3 (1992), pp. 448–472.

[187] Reza Mahjourian, Martin Wicke, and Anelia Angelova. “Unsupervised Learning of Depth and Ego-Motion from Monocular Video Using 3D Geometric Constraints”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2018.

[188] S.M.S.D. Malleswari and T. Krishna Mohana. "Air pollution monitoring system using IoT devices: Review". In: *Materials Today: Proceedings* (2022).

[189] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[190] Larry Matthies. "Stereo vision for planetary rovers: Stochastic modeling to near real-time implementation". In: *International Journal of Computer Vision* 8.1 (1992), pp. 71–91.

[191] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation". In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2016.

[192] Robert McCraith, Lukas Neumann, Andrew Zisserman, and Andrea Vedaldi. "Monocular Depth Estimation with Self-supervised Instance Adaptation". In: *arXiv preprint arXiv:2004.05821* (2020).

[193] Max Mehltretter and Christian Heipke. "Cnn-based cost volume analysis as confidence measure for dense matching". In: *International Conference on Computer Vision Workshops*. IEEE, 2019.

[194] Simon Meister, Junhwa Hur, and Stefan Roth. "UnFlow: Unsupervised Learning of Optical Flow with a Bidirectional Census Loss". In: *AAAI Conference on Artificial Intelligence*. 2018.

[195] Étienne Mémin and Pepito Perez. "A multigrid approach for hierarchical motion estimation". In: Feb. 1998, pp. 933–938.

[196] Moritz Menze and Andreas Geiger. "Object Scene Flow for Autonomous Vehicles". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2015.

[197] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *European Conference on Computer Vision*. 2020.

[198] Yana Mileva, Andrés Bruhn, and Joachim Weickert. "Illumination-robust variational optical flow with photometric invariants". In: *Joint Pattern Recognition Symposium*. Springer. 2007, pp. 152–162.

[199] Andres Milioto, Philipp Lottes, and Cyrill Stachniss. "Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in CNNs". In: *International Conference on Robotics and Automation*. IEEE. 2018.

[200] Shervin Minaee, Yuri Y Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. "Image segmentation using deep learning: A survey". In: *Transactions on Pattern Analysis and Machine Intelligence* (2021).

[201] Alessio Mingozzi, Andrea Conti, Filippo Aleotti, Matteo Poggi, and Stefano Mattoccia. "Monitoring social distancing with single image depth estimation". In: (2021). IEEE Transactions on Emerging Topics in Computational Intelligence.

[202] Eduardo F Morales, Rafael Murrieta-Cid, Israel Becerra, and Marco A Esquivel-Basaldua. "A survey on deep learning and deep reinforcement learning in robotics with a tutorial on deep reinforcement learning". In: *Intelligent Service Robotics* (2021).

[203] Mohsen Mousavi and Amir H. Gandomi. "Deep learning for structural health monitoring under environmental and operational variations". In: *Nondestructive Characterization and Monitoring of Advanced Materials, Aerospace, Civil Infrastructure, and Transportation XV*. 2021.

[204] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding". In: (2022).

[205] Raúl Mur-Artal and Juan D. Tardós. "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras". In: *IEEE Transactions on Robotics* (2017).

[206] Pushmeet Kohli Nathan Silberman Derek Hoiem and Rob Fergus. "Indoor Segmentation and Support Inference from RGBD Images". In: *European Conference on Computer Vision*. Springer. 2012.

[207] Radford M Neal. *Bayesian learning for neural networks*. Vol. 118. Springer Science & Business Media, 2012.

[208] Cong T Nguyen, Yuris Mulya Saputra, Nguyen Van Huynh, Ngoc-Tan Nguyen, Tran Viet Khoa, Bui Minh Tuan, Diep N Nguyen, Dinh Thai Hoang, Thang X Vu, Eryk Dutkiewicz, et al. "A comprehensive survey of enabling and emerging technologies for social distancing—Part I: Fundamentals and enabling technologies". In: *IEEE Access* 8 (2020), pp. 153479–153507.

[209] Cong T Nguyen, Yuris Mulya Saputra, Nguyen Van Huynh, Ngoc-Tan Nguyen, Tran Viet Khoa, Bui Minh Tuan, Diep N Nguyen, Dinh Thai Hoang, Thang X Vu, Eryk Dutkiewicz, et al. "A comprehensive survey of enabling and emerging technologies for social distancing—Part II: Emerging technologies and open issues". In: *IEEE Access* 8 (2020), pp. 154209–154236.

[210] FuTao Ni, Jian Zhang, and ZhiQiang Chen. "Pixel-level crack delineation in images with convolutional feature fusion". In: *Structural Control and Health Monitoring* (2019), e2286.

[211] Michael Niemeyer, Jonathan T. Barron, Ben Mildenhall, Mehdi S. M. Sajjadi, Andreas Geiger, and Noha Radwan. "RegNeRF: Regularizing Neural Radiance Fields for View Synthesis from Sparse Inputs". In: *Conference on Computer Vision and Pattern Recognition*. 2022.

[212] David A Nix and Andreas S Weigend. "Estimating the mean and variance of the target probability distribution". In: *International Conference on Neural Networks*. Vol. 1. IEEE. 1994.

[213] A.S. Ogale and Y. Aloimonos. "Stereo correspondence with slanted surfaces: critical implications of horizontal slant". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2004.

[214] Mahdi Maktab Dar Oghaz, Manzoor Razaak, Hamideh Kerdegari, Vasileios Argyriou, and Paolo Remagnino. "Scene and environment monitoring using aerial imagery and deep learning". In: *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE. 2019.

[215] Masatoshi Okutomi and Takeo Kanade. "A locally adaptive window for signal matching". In: *International Journal of Computer Vision* 7.2 (1992), pp. 143–162.

[216] Art B Owen. "A robust hybrid of lasso and ridge regression". In: *Contemporary Mathematics* 443.7 (2007), pp. 59–72.

[217] Jiahao Pang, Wenxiu Sun, Jimmy SJ Ren, Chengxi Yang, and Qiong Yan. "Cascade residual learning: A two-stage convolutional neural network for stereo matching". In: *International Conference on Computer Vision Workshops*. IEEE. 2017.

[218] Nils Papenberg, Andres Bruhn, Thomas Brox, Stephan Didas, and Joachim Weickert. "Highly Accurate Optic Flow Computation with Theoretically Justified Warping". In: *International Journal of Computer Vision* 67 (Apr. 2006), pp. 141–158.

[219] Haesol Park and Kyoung Mu Lee. "Look wider to match image patches with convolutional neural networks". In: *Signal Processing Letters* 24.12 (2016), pp. 1788–1792.

[220] Min-Gyu Park and Kuk-Jin Yoon. "Learning and selecting confidence measures for robust stereo matching". In: *Transactions on Pattern Analysis and Machine Intelligence* 41.6 (2018), pp. 1397–1411.

[221] Min-Gyu Park and Kuk-Jin Yoon. "Leveraging stereo matching with learning-based confidence measures". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2015.

[222] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019.

[223] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung. “A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2016.

[224] Husein Perez, Joseph HM Tah, and Amir Mosavi. “Deep learning for detecting building defects using convolutional neural networks”. In: *Sensors* (2019).

[225] Pietro Perona and Jitendra Malik. “Scale-space and edge detection using anisotropic diffusion”. In: *Transactions on Pattern Analysis and Machine Intelligence* 12.7 (1990), pp. 629–639.

[226] Andrea Pilzer, Stephane Lathuiliere, Nicu Sebe, and Elisa Ricci. “Refine and Distill: Exploiting Cycle-Inconsistency and Knowledge Distillation for Unsupervised Monocular Depth Estimation.” In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[227] Pedro O Pinheiro, Ronan Collobert, and Piotr Dollár. “Learning to segment object candidates”. In: *arXiv preprint arXiv:1506.06204* (2015).

[228] Nils Plath, Marc Toussaint, and Shinichi Nakajima. “Multi-class image segmentation using conditional random fields and global classification”. In: *International Conference on Machine Learning*. 2009.

[229] Matteo Poggi, Filippo Aleotti, Fabio Tosi, and Stefano Mattoccia. “On the uncertainty of self-supervised monocular depth estimation”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2020.

[230] Matteo Poggi, Filippo Aleotti, Fabio Tosi, and Stefano Mattoccia. “Towards real-time unsupervised monocular depth estimation on CPU”. In: *International Conference on Intelligent Robots and Systems*. 2018.

[231] Matteo Poggi, Seungryong Kim, Fabio Tosi, Sunok Kim, Filippo Aleotti, Dongbo Min, Kwanghoon Sohn, and Stefano Mattoccia. “On the confidence of stereo matching in a deep-learning era: a quantitative evaluation”. In: *Transactions on Pattern Analysis and Machine Intelligence* (2021).

[232] Matteo Poggi and Stefano Mattoccia. “Learning a general-purpose confidence measure based on o (1) features and a smarter aggregation strategy for semi global matching”. In: *International Conference on 3D Vision*. IEEE. 2016.

[233] Matteo Poggi and Stefano Mattoccia. “Learning from scratch a confidence measure”. In: *British Machine Vision Conference*. BMVA, 2016.

[234] Matteo Poggi, Fabio Tosi, Filippo Aleotti, and Stefano Mattoccia. “Real-time Self-Supervised Monocular Depth Estimation Without GPU”. In: (2021). IEEE Transactions on Intelligent Transportation Systems.

[235] Matteo Poggi, Fabio Tosi, Konstantinos Batsos, Philippos Mordohai, and Stefano Mattoccia. “On the Synergies between Machine Learning and Binocular Stereo for Depth Estimation from Images: a Survey”. In: *Transactions on Pattern Analysis and Machine Intelligence* (2021).

[236] Matteo Poggi, Fabio Tosi, and Stefano Mattoccia. “Learning a confidence measure in the disparity domain from O (1) features”. In: *Computer Vision and Image Understanding* 193 (2020), p. 102905.

[237] Matteo Poggi, Fabio Tosi, and Stefano Mattoccia. “Learning monocular depth estimation with unsupervised trinocular assumptions”. In: *International Conference on 3D Vision*. 2018.

[238] Matteo Poggi, Fabio Tosi, and Stefano Mattoccia. “Quantitative evaluation of confidence measures in a machine learning world”. In: *International Conference on Computer Vision*. IEEE, 2017.

[239] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. "D-nerf: Neural radiance fields for dynamic scenes". In: *Conference on Computer Vision and Pattern Recognition*. 2021.

[240] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. "Learning transferable visual models from natural language supervision". In: *International Conference on Machine Learning*. PMLR. 2021.

[241] Chakravartula Raghavachari, V. Aparna, S. Chithira, and Vidhya Balasubramanian. "A Comparative Study of Vision Based Human Detection Techniques in People Counting Applications". In: *Procedia Computer Science* 58 (2015). International Symposium on Computer Vision and the Internet, pp. 461–469. ISSN: 1877-0509.

[242] Michael Ramamonjisoa, Michael Firman, Jamie Watson, Vincent Lepetit, and Daniyar Turmukhambetov. "Single Image Depth Prediction With Wavelet Decomposition". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2021.

[243] Xukan Ran, Haolianz Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. "Deepdecision: A mobile deep learning framework for edge video analytics". In: *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE. 2018, pp. 1421–1429.

[244] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. "Vision transformers for dense prediction". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2021.

[245] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. "Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer". In: *Transactions on Pattern Analysis and Machine Intelligence* (2020).

[246] Anurag Ranjan and Michael J Black. "Optical flow estimation using a spatial pyramid network". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2017.

[247] Carlo S Regazzoni, Andrea Cavallaro, Ying Wu, Janusz Konrad, and Arun Hampapur. "Video analytics for surveillance: Theory and practice [from the guest editors]". In: *IEEE Signal Processing Magazine* 27.5 (2010), pp. 16–17.

[248] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. "Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps". In: *International Conference on Computer Vision*. 2021.

[249] Zhile Ren, Orazio Gallo, Deqing Sun, Ming-Hsuan Yang, Erik B Sudderth, and Jan Kautz. "A fusion approach for multi-frame optical flow estimation". In: *Winter Conference on Applications of Computer Vision*. IEEE. 2019.

[250] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. "EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2015.

[251] Barbara Roessle, Jonathan T. Barron, Ben Mildenhall, Pratul P. Srinivasan, and Matthias Nießner. "Dense Depth Priors for Neural Radiance Fields from Sparse Input Views". In: *Conference on Computer Vision and Pattern Recognition*. 2022.

[252] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2015.

[253] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. "The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2016.

[254] Stefan Roth and Michael J Black. "On the spatial statistics of optical flow". In: *International Journal of Computer Vision* 74.1 (2007), pp. 33–50.

[255] Daniel Rueckert, Luke I Sonoda, Carmel Hayes, Derek LG Hill, Martin O Leach, and David J Hawkes. "Nonrigid registration using free-form deformations: application to breast MR images". In: *Transactions on Medical Imaging* 18.8 (1999), pp. 712–721.

[256] Mohd Ezanee Rusli, Salman Yussof, Mohammad Ali, and Ahmed Abdullah Abobakr Hassan. “Mysd: a smart social distancing monitoring system”. In: *International Conference on Information Technology and Multimedia*. IEEE. 2020, pp. 399–403.

[257] Sadra Safadoust and Fatma Güney. “Self-Supervised Monocular Scene Decomposition and Depth Estimation”. In: *International Conference on 3D Vision*. IEEE. 2021.

[258] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. “PIFu: Pixel-Aligned Implicit Function for High-Resolution Clothed Human Digitization”. In: *International Conference on Computer Vision*. IEEE, 2019.

[259] Seyed Omid Sajedi and Xiao Liang. “A convolutional cost-sensitive crack localization algorithm for automated and reliable RC bridge inspection”. In: *Risk-Based Bridge Engineering: Proceedings of the 10th New York City Bridge Conference*. 2019.

[260] Marouane Salhaoui, J Carlos Molina-Molina, Antonio Guerrero-González, Mounir Arioua, and Francisco J Ortiz. “Autonomous underwater monitoring system for detecting life on the seabed by means of computer vision cloud services”. In: *Remote Sensing* (2020).

[261] Ashutosh Saxena, Sung H Chung, Andrew Y Ng, et al. “Learning depth from single monocular images”. In: *Conference on Neural Information Processing Systems*. 2005.

[262] Ashutosh Saxena, Min Sun, and Andrew Y Ng. “Make3d: Learning 3d scene structure from a single still image”. In: *IEEE Transaction on Pattern Analysis and Machine Intelligence* 31.5 (2009), pp. 824–840.

[263] Daniel Scharstein, Heiko Hirschmüller, York Kitajima, Greg Krathwohl, Nera Nešić, Xi Wang, and Porter Westling. “High-resolution stereo datasets with subpixel-accurate ground truth”. In: *German Conference on Pattern Recognition*. Springer. 2014.

[264] Daniel Scharstein and Richard Szeliski. “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms”. In: *International Journal of Computer Vision* 47.1-3 (2002), pp. 7–42.

[265] Daniel Scharstein and Richard Szeliski. “Stereo matching with nonlinear diffusion”. In: *International Journal of Computer Vision* 28.2 (1998), pp. 155–174.

[266] Johannes L Schonberger, Sudipta N Sinha, and Marc Pollefeys. “Learning to fuse proposals from multiple scanline optimizations in semi-global matching”. In: *European Conference on Computer Vision*. Springer. 2018, pp. 739–755.

[267] Johannes Lutz Schönberger and Jan-Michael Frahm. “Structure-from-Motion Revisited”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2016.

[268] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. “Pixelwise View Selection for Unstructured Multi-View Stereo”. In: *European Conference on Computer Vision*. Springer. 2016.

[269] Thomas Schops, Johannes L Schonberger, Silvano Galliani, Torsten Sattler, Konrad Schindler, Marc Pollefeys, and Andreas Geiger. “A multi-view stereo benchmark with high-resolution images and multi-camera videos”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2017.

[270] Florian Schroff, Antonio Criminisi, and Andrew Zisserman. “Object Class Segmentation using Random Forests.” In: *British Machine Vision Conference*. 2008.

[271] Akihito Seki and Marc Pollefeys. “Sgm-nets: Semi-global matching with neural networks”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2017.

[272] Andrew W Senior, L Brown, Arun Hampapur, C-F Shu, Yun Zhai, Rogério Schmidt Feris, Y-L Tian, Sergio Borger, and C Carlson. “Video analytics for retail”. In: *Conference on Advanced Video and Signal Based Surveillance*. IEEE. 2007, pp. 423–428.

[273] Amit Shaked and Lior Wolf. “Improved stereo matching with constant highway networks and reflective confidence learning”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2017.

[274] Zhenfeng Shao, Gui Cheng, Jiayi Ma, Zhongyuan Wang, Jiaming Wang, and Deren Li. “Real-time and Accurate UAV Pedestrian Detection for Social Distancing Monitoring in COVID-19 Pandemic”. In: *Transactions on Multimedia* (2021).

[275] Meng-Li Shih, Shih-Yang Su, Johannes Kopf, and Jia-Bin Huang. “3D Photography using Context-aware Layered Depth Inpainting”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2020.

[276] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: (2014).

[277] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. “Implicit Neural Representations with Periodic Activation Functions”. In: (2020).

[278] Xiao Song, Xu Zhao, Liangji Fang, Hanwen Hu, and Yizhou Yu. “Edgestereo: An effective multi-task learning network for stereo matching and edge detection”. In: *International Journal of Computer Vision* 128.4 (2020), pp. 910–930.

[279] Xiao Song, Xu Zhao, Hanwen Hu, and Liangji Fang. “Edgestereo: A context integrated residual pyramid network for stereo matching”. In: *Asian Conference on Computer Vision*. Springer. 2018.

[280] Robert Spangenberg, Tobias Langner, Sven Adfeldt, and Raúl Rojas. “Large scale semi-global matching on the cpu”. In: *Intelligent Vehicles Symposium*. IEEE. 2014, pp. 195–201.

[281] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[282] Sriyanka and Siddarama R Patil. “Smart Environmental Monitoring through Internet of Things (IoT) using RaspberryPi 3”. In: *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*. 2017.

[283] Austin Stone, Daniel Maurer, Alper Ayvaci, Anelia Angelova, and Rico Jonschkowski. “SMURF: Self-Teaching Multi-Frame Unsupervised RAFT with Full-Image Warping”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2021.

[284] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. “A benchmark for the evaluation of RGB-D SLAM systems”. In: *International Conference on Intelligent Robots and Systems*. IEEE. 2012.

[285] Deqing Sun, Stefan Roth, and Michael J Black. “A quantitative analysis of current practices in optical flow estimation and the principles behind them”. In: *International Journal of Computer Vision* 106.2 (2014), pp. 115–137.

[286] Deqing Sun, Stefan Roth, J.P. Lewis, and Michael Black. “Learning Optical Flow”. In: *European Conference on Computer Vision*. Vol. 1. Springer. 2008.

[287] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. “PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2018.

[288] Qiyu Sun, Gary G Yen, Yang Tang, and Chaoqiang Zhao. “Learn to Adapt for Monocular Depth Estimation”. In: *arXiv preprint arXiv:2203.14005* (2022).

[289] Joakim Svensk. *Evaluation of aerial image stereo matching methods for forest variable estimation*. 2017.

[290] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. “Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains”. In: 2020.

[291] Vladimir Tankovich, Christian Hane, Yinda Zhang, Adarsh Kowdle, Sean Fanello, and Sofien Bouaziz. “HITNet: Hierarchical Iterative Tile Refinement Network for Real-time Stereo Matching”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2021.

[292] Flavia Tauro, Fabio Tosi, Stefano Mattoccia, Elena Toth, Rodolfo Piscopia, and Salvatore Grimaldi. In: *Remote Sensing* 10.12 (2018).

[293] Graham W Taylor, Rob Fergus, Yann LeCun, and Christoph Bregler. “Convolutional learning of spatio-temporal features”. In: *European Conference on Computer Vision*. Springer. 2010.

[294] Zachary Teed and Jia Deng. “RAFT: Recurrent All-Pairs Field Transforms for Optical Flow”. In: *European Conference on Computer Vision*. Springer. 2020.

[295] Alexandru Telea. “An image inpainting technique based on the fast marching method”. In: *Journal of graphics tools* 9.1 (2004), pp. 23–34.

[296] Lokender Tiwari, Pan Ji, Quoc-Huy Tran, Bingbing Zhuang, Saket Anand, and Manmohan Chandraker. “Pseudo RGB-D for Self-Improving Monocular SLAM and Depth Prediction”. In: *European Conference on Computer Vision*. 2020.

[297] Marco Toldo, Andrea Maracani, Umberto Michieli, and Pietro Zanuttigh. “Unsupervised Domain Adaptation in Semantic Segmentation: a Review”. In: *arXiv preprint arXiv:2005.10876* (2020).

[298] Federico Tombari, Luigi Di Stefano, Stefano Mattoccia, and Angelo Galanti. “Performance Evaluation of Robust Matching Measures.” In: vol. 1. Jan. 2008, pp. 473–478.

[299] Federico Tombari, Stefano Mattoccia, Luigi Di Stefano, and Elisa Addimanda. “Classification and evaluation of cost aggregation methods for stereo correspondence”. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8.

[300] Alessio Tonioni, Matteo Poggi, Stefano Mattoccia, and Luigi Di Stefano. “Unsupervised Adaptation for Deep Stereo”. In: *International Conference on Computer Vision*. IEEE, 2017.

[301] Alessio Tonioni, Fabio Tosi, Matteo Poggi, Stefano Mattoccia, and Luigi Di Stefano. “Real-Time Self-Adaptive Deep Stereo”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[302] Fabio Tosi, Filippo Aleotti, Matteo Poggi, and Stefano Mattoccia. “Learning monocular depth estimation infusing traditional stereo knowledge”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[303] Fabio Tosi, Yiyi Liao, Carolin Schmitt, and Andreas Geiger. “SMD-Nets: Stereo Mixture Density Networks”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2021.

[304] Fabio Tosi, Matteo Poggi, Antonio Benincasa, and Stefano Mattoccia. “Beyond local reasoning for stereo confidence estimation with deep learning”. In: *European Conference on Computer Vision*. Springer. 2018.

[305] Fabio Tosi, Matteo Poggi, Alessio Tonioni, Luigi Di Stefano, and Stefano Mattoccia. “Learning confidence measures in the wild”. In: *British Machine Vision Conference*. BMVA, 2017.

[306] Vlad Trifa, Lewis Girod, Travis Collier, Daniel Blumstein, Charles Taylor, Atr Cns, Humanoid Robotics, Comp Science, and Ai Lab. “Automated wildlife monitoring using self-configuring sensor networks deployed in natural habitats”. In: (2007).

[307] Stepan Tulyakov, Anton Ivanov, and Francois Fleuret. “Weakly supervised learning of deep metrics for stereo reconstruction”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2017.

[308] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. “Sparsity Invariant CNNs”. In: *International Conference on 3D Vision*. IEEE, 2017.

[309] Igor Vasiljevic, Vitor Guizilini, Rares Ambrus, Sudeep Pillai, Wolfram Burgard, Greg Shakhnarovich, and Adrien Gaidon. "Neural Ray Surfaces for Self-Supervised Learning of Depth and Ego-motion". In: *International Conference on 3D Vision*. IEEE. 2020.

[310] Igor Vasiljevic et al. "DIODE: A Dense Indoor and Outdoor DEpth Dataset". In: *Computing Research Repository* abs/1908.00463 (2019).

[311] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Conference on Neural Information Processing Systems* (2017).

[312] Olga Veksler. "Stereo matching by compact windows via minimum ratio cycle". In: *International Conference on Computer Vision*. IEEE. 2001.

[313] Christoph Vogel, Stefan Roth, and Konrad Schindler. "An evaluation of data costs for optical flow". In: *German Conference on Pattern Recognition*. Springer. 2013.

[314] Chaoyang Wang, Jose Miguel Buenaposada, Rui Zhu, and Simon Lucey. "Learning depth from monocular videos using direct methods". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2018.

[315] Han-Yang Wang, Ya-Ching Chang, Yi-Yu Hsieh, Hua-Tsung Chen, and Jen-Hui Chuang. "Deep learning-based human activity analysis for aerial images". In: *2017 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*. 2017.

[316] Hengli Wang, Rui Fan, and Ming Liu. "Co-Teaching: An Ark to Unsupervised Stereo Matching". In: *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2021, pp. 3328–3332.

[317] Yan Wang, Zihang Lai, Gao Huang, Brian H Wang, Laurens Van Der Maaten, Mark Campbell, and Kilian Q Weinberger. "Anytime stereo image depth estimation on mobile devices". In: *International Conference on Robotics and Automation*. IEEE. 2019.

[318] Yang Wang, Peng Wang, Zhenheng Yang, Chenxu Luo, Yi Yang, and Wei Xu. "UnOS: Unified Unsupervised Optical-Flow and Stereo-Depth Estimation by Watching Videos". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[319] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. "Image quality assessment: from error visibility to structural similarity". In: *Transactions on Image Processing* 13.4 (2004), pp. 600–612.

[320] Jamie Watson, Oisin Mac Aodha, Daniyar Turmukhambetov, Gabriel J. Brostow, and Michael Firman. "Learning Stereo from Single Images". In: *European Conference on Computer Vision*. Springer. 2020.

[321] Jamie Watson, Michael Firman, Gabriel J Brostow, and Daniyar Turmukhambetov. "Self-Supervised Monocular Depth Hints". In: *International Conference on Computer Vision*. IEEE, 2019.

[322] Jamie Watson, Oisin Mac Aodha, Victor Prisacariu, Gabriel Brostow, and Michael Firman. "The Temporal Opportunist: Self-Supervised Multi-Frame Monocular Depth". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2021.

[323] Mark Weber, Jonathon Luiten, and Bastian Leibe. "Single-shot panoptic segmentation". In: *International Conference on Intelligent Robots and Systems*. IEEE. 2020.

[324] Yi Wei, Shaohui Liu, Yongming Rao, Wang Zhao, Jiwen Lu, and Jie Zhou. "Nerfingmvs: Guided optimization of neural radiance fields for indoor multi-view stereo". In: *International Conference on Computer Vision*. 2021.

[325] Manuel Werlberger, Thomas Pock, Markus Unger, and Horst Bischof. "Optical flow guided TV-L 1 video interpolation and restoration". In: *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*. Springer. 2011.

[326] Diana Wofk, Fangchang Ma, Tien-Ju Yang, Sertac Karaman, and Vivienne Sze. "Fast-Depth: Fast Monocular Depth Estimation on Embedded Systems". In: *International Conference on Robotics and Automation*. IEEE, 2019.

[327] Alex Wong and Stefano Soatto. "Bilateral cyclic constraint and adaptive regularization for unsupervised monocular depth prediction". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[328] Chih-Hsuan Wu, Jun-Wei Hsieh, Chia-Yu Wang, and Chih-Hsiang Ho. "Marine Pollution Detection based on Deep Learning and Optical Flow". In: *2020 International Computer Symposium (ICS)*. IEEE. 2020.

[329] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. *Detectron2*. 2019.

[330] Marie Xavier, Alain Lalande, Paul M Walker, François Brunotte, and Louis Legrand. "An adapted optical flow algorithm for robust quantification of cardiac wall motion from standard cine-MR examinations". In: *Transactions on Information Technology in Biomedicine* 16.5 (2012), pp. 859–868.

[331] Ke Xian, Jianming Zhang, Oliver Wang, Long Mai, Zhe Lin, and Zhiguo Cao. "Structure-guided ranking loss for single image depth prediction". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2020.

[332] Junyuan Xie, Ross Girshick, and Ali Farhadi. "Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks". In: *European Conference on Computer Vision*. Springer. 2016.

[333] Yuwen Xiong, Renjie Liao, Hengshuang Zhao, Rui Hu, Min Bai, Ersin Yumer, and Raquel Urtasun. "Upsnet: A unified panoptic segmentation network". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[334] Haofei Xu and Juyong Zhang. "AANet: Adaptive Aggregation Network for Efficient Stereo Matching". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2020.

[335] Haofei Xu, Jianmin Zheng, Jianfei Cai, and Juyong Zhang. "Region Deformer Networks for Unsupervised Depth Estimation from Unconstrained Monocular Videos". In: *IJCAI*. 2019.

[336] Li Xu, Jiaya Jia, and Yasuyuki Matsushita. "Motion detail preserving optical flow estimation". In: *Transactions on Pattern Analysis and Machine Intelligence* 34.9 (2011), pp. 1744–1757.

[337] Gengshan Yang, Joshua Manela, Michael Happold, and Deva Ramanan. "Hierarchical deep stereo matching on high-resolution images". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[338] Guorun Yang, Xiao Song, Chaoqin Huang, Zhidong Deng, Jianping Shi, and Bolei Zhou. "DrivingStereo: A Large-Scale Dataset for Stereo Matching in Autonomous Driving Scenarios". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[339] Guorun Yang, Hengshuang Zhao, Jianping Shi, Zhidong Deng, and Jiaya Jia. "Seg-Stereo: Exploiting Semantic Information for Disparity Estimation". In: *European Conference on Computer Vision*. Springer, 2018.

[340] Nan Yang, Lukas von Stumberg, Rui Wang, and Daniel Cremers. "D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2020.

[341] Nan Yang, Rui Wang, Jörg Stückler, and Daniel Cremers. "Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry". In: *European Conference on Computer Vision*. Springer. 2018.

[342] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. "Netadapt: Platform-aware neural network adaptation for mobile applications". In: *European Conference on Computer Vision*. Springer, 2018.

[343] Zhenheng Yang, Peng Wang, Yang Wang, Wei Xu, and Ram Nevatia. "Every Pixel Counts: Unsupervised Geometry Learning with Holistic 3D Motion Understanding". In: *European Conference on Computer Vision Workshops*. Springer. 2018.

[344] XW Ye, T Jin, and CB Yun. "A review on deep learning-based structural health monitoring of civil infrastructures". In: *Smart Structures and Systems* (2019).

[345] Wei Yin, Yifan Liu, Chunhua Shen, and Youliang Yan. "Enforcing geometric constraints of virtual normal for depth prediction". In: *International Conference on Computer Vision*. IEEE, 2019.

[346] Wei Yin, Jianming Zhang, Oliver Wang, Simon Niklaus, Long Mai, Simon Chen, and Chunhua Shen. "Learning to recover 3d scene shape from a single image". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2021.

[347] Zhichao Yin and Jianping Shi. "GeoNet: Unsupervised Learning of Dense Depth, Optical Flow and Camera Pose". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2018.

[348] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. "pixelNeRF: Neural Radiance Fields from One or Few Images". In: *Conference on Computer Vision and Pattern Recognition*. 2021.

[349] Zehao Yu, Lei Jin, and Shenghua Gao. "P^ 2 2 Net: Patch-Match and Plane-Regularization for Unsupervised Indoor Depth Estimation". In: *European Conference on Computer Vision*. Springer. 2020.

[350] Qiangqiang Yuan, Huanfeng Shen, Tongwen Li, Zhiwei Li, Shuwen Li, Yun Jiang, Hongzhang Xu, Weiwei Tan, Qianqian Yang, Jiwen Wang, et al. "Deep learning in environmental remote sensing: Achievements and challenges". In: *Remote Sensing of Environment* (2020).

[351] Ramin Zabih and John Woodfill. "Non-parametric Local Transforms for Computing Visual Correspondence". In: *European Conference on Computer Vision*. Springer, 1994, pp. 151–158.

[352] Pierluigi Zama Ramirez, Matteo Poggi, Fabio Tosi, Stefano Mattoccia, and Luigi Di Stefano. "Geometry meets semantic for semi-supervised monocular depth estimation". In: *Asian Conference on Computer Vision*. 2018.

[353] Jure Zbontar and Yann LeCun. "Stereo matching by training a convolutional neural network to compare image patches". In: *Journal of Machine Learning Research* 17.1-32 (2016), p. 2.

[354] Feihu Zhang, Victor Prisacariu, Ruigang Yang, and Philip HS Torr. "GA-Net: Guided Aggregation Net for End-to-end Stereo Matching". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[355] Feihu Zhang, Xiaojuan Qi, Ruigang Yang, Victor Prisacariu, Benjamin Wah, and Philip Torr. "Domain-invariant Stereo Matching Networks". In: *European Conference on Computer Vision*. Springer, 2020.

[356] Zhenyu Zhang, Stéphane Lathuilière, Elisa Ricci, Nicu Sebe, Yan Yan, and Jian Yang. "Online depth learning against forgetting in monocular videos". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2020.

[357] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. "Pyramid scene parsing network". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2017.

[358] Wang Zhao, Shaohui Liu, Yezhi Shu, and Yong-Jin Liu. "Towards Better Generalization: Joint Depth-Pose Learning without PoseNet". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2020.

[359] Yu Zhao, Quan Chen, Wengang Cao, Jie Yang, Jian Xiong, and Guan Gui. "Deep learning for risk detection and trajectory tracking at construction sites". In: *IEEE Access* (2019).

[360] Mengyu Zheng, Chuan Zhou, Jia Wu, and Li Guo. “Smooth Deep Network Embedding”. In: *International Joint Conference on Neural Networks*. IEEE. 2019.

[361] Yiran Zhong, Hongdong Li, and Yuchao Dai. “Open-world stereo video matching with deep rnn”. In: *European Conference on Computer Vision*. Springer, 2018.

[362] Yiran Zhong, Hongdong Li, and Yuchao Dai. “Self-supervised learning for stereo matching with self-improving ability.” In: *arXiv preprint arXiv:1709.00930* (2017).

[363] Chao Zhou, Hong Zhang, Xiaoyong Shen, and Jiaya Jia. “Unsupervised Learning of Stereo Matching”. In: *International Conference on Computer Vision*. IEEE, 2017.

[364] Junsheng Zhou, Yuwang Wang, Naiyan Wang, and Wenjun Zeng. “Unsupervised High-Resolution Depth Learning from Videos with Dual Networks”. In: *International Conference on Computer Vision*. IEEE. IEEE, 2019.

[365] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. “Unsupervised learning of depth and ego-motion from video”. In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2017.

[366] Federica Zonzini, Denis Bogomolov, Tanush Dhamija, Nicola Testoni, Luca De Marchi, and Alessandro Marzani. “Deep Learning Approaches for Robust Time of Arrival Estimation in Acoustic Emission Monitoring”. In: *Sensors* 22 (2022).

[367] Yuliang Zou, Zelun Luo, and Jia-Bin Huang. “DF-Net: Unsupervised Joint Learning of Depth and Flow using Cross-Task Consistency”. In: *European Conference on Computer Vision*. Springer. 2018.