

**ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA**

---

**DEPARTMENT OF COMPUTER SCIENCE  
AND ENGINEERING**

**ARTIFICIAL INTELLIGENCE**

**MASTER THESIS**

in

Sistemi Digitali M

**Realtime Monocular Depth Estimation on Mobile  
Phones**

**CANDIDATE**

Marco Rovinelli

**SUPERVISOR**

Prof. Stefano Mattocchia

**CO-SUPERVISORS**

Dr. Matteo Poggi

Dr. Alessandro Pieropan

Eng. Lorenzo Andraghetti

Eng. Fereidoon Zangeneh

**Academic Year 2020/21**

**Session II**

To my family

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Depth maps . . . . .	5
1.2	Active sensors . . . . .	6
1.3	Image-based depth estimation . . . . .	6
1.3.1	Learning depth estimation . . . . .	7
1.4	Self-supervised training . . . . .	8
1.5	Why mobile phones as target platform . . . . .	9
1.6	Thesis Structure . . . . .	9
<b>2</b>	<b>Related Work</b>	<b>10</b>
2.1	Supervised Monocular Depth Estimation . . . . .	10
2.2	Self-Supervised Monocular Depth Estimation . . . . .	12
2.3	Compact architectures for real-time performance . . . . .	13
2.4	SLAM and Visual Odometry . . . . .	15
<b>3</b>	<b>The Baseline</b>	<b>16</b>
3.1	Depth estimation as Image Reconstruction . . . . .	17
3.2	Network Structure . . . . .	18
3.3	Losses . . . . .	20
3.4	Experimental Setup . . . . .	21

3.4.1	KITTI Dataset . . . . .	21
3.4.2	Stockholm Dataset . . . . .	22
3.4.3	Depth Metrics . . . . .	23
3.4.4	Depth from object size . . . . .	24
3.5	Pose Metrics . . . . .	26
3.6	Implementation Details . . . . .	27
<b>4</b>	<b>Experimentation</b>	<b>29</b>
4.1	Faster Neural Networks . . . . .	29
4.1.1	Encoders . . . . .	30
4.1.2	Decoders . . . . .	33
4.1.3	Pose Networks . . . . .	35
4.2	Sparse Depths . . . . .	37
4.2.1	Masking Sparse depths . . . . .	41
4.3	Scale recovery . . . . .	42
4.4	Quantization . . . . .	44
4.5	Additional approaches tried . . . . .	47
4.5.1	Disparity Probability Volumes . . . . .	47
4.5.2	Temporal Consistency . . . . .	49
4.5.3	YUV input data . . . . .	50
<b>5</b>	<b>Evaluation and Results</b>	<b>52</b>
5.1	KITTI Dataset Results . . . . .	52
5.2	Stockholm Dataset . . . . .	53
5.3	Pose Evaluation . . . . .	54
<b>6</b>	<b>Conclusions and further developments</b>	<b>58</b>
	<b>References</b>	<b>60</b>

# Abstract

Depth estimation is a necessary task to understand and navigate the environment around us. Over the years, many active sensors have been developed to measure depth but they are expensive and require additional space to be mounted. A cheaper alternative consists of estimating depth maps using images taken by a mobile phone camera. Since most mobile phones don't have cameras built for stereo depth sensing, it would be ideal to be able to recover depth from a single image using only the computational capability of the mobile phone itself. This can be achieved by training a neural network on ground truth depth maps. This type of data is very expensive to obtain so it's preferred to train the neural network using self-supervision from multiple images. Since the devices where the trained models will be deployed have only one camera, it is ideal to train the network on monocular videos representing the actual data distribution at deployment. Self-supervised training using monocular videos lowers the accuracy of the depth maps and brings the additional challenge of being able to predict depth only up to an unknown scale factor. To this end, additional information, velocity provided by the GPS, and sparse points computed by a monocular SLAM algorithm, are employed to recover scale and improve the accuracy. This study will investigate different neural network architectures and training schemes to achieve depth maps as accurately as possible given the constraints of the computational budget available on modern mobile phones.

# Chapter 1

## Introduction

### 1.1 Depth maps

A depth map is a single-channel image where each pixel value corresponds to the distance of the viewed object from the camera center along the principal axis. This type of information is necessary for an agent to create an internal three-dimensional reconstruction of the scene. Applications of depth estimation range from autonomous driving where vehicles need depth estimation to understand how far away other objects are, to augmented reality where a 3D scene reconstruction is needed to properly visualize virtual objects, to photography where depth information is used to simulate bokeh effects.

## 1.2 Active sensors

Depth maps can be acquired through several techniques and tools. Some of these rely on active sensors such as LIDARs and can achieve very high accuracy in ideal conditions. This type of sensor uses the time of flight of a laser beam to measure depth. By rotating the laser and casting many beams at different angles, it can generate sparse depth maps such as the ones in Figure 1.1. However, LIDARs are expensive and require a lot of mounting space, this makes it unrealistic to mount one on every agent that needs depth-sensing capabilities. Moreover, LIDARs struggle to provide correct estimates for moving and reflective objects. Other active light-based sensors like Microsoft Kinect suffer from limited depth estimation range. RADARs can also provide depth information but it struggles with bridges and its depth maps are very sparse that makes them unsuitable for any accurate 3D scene reconstruction.

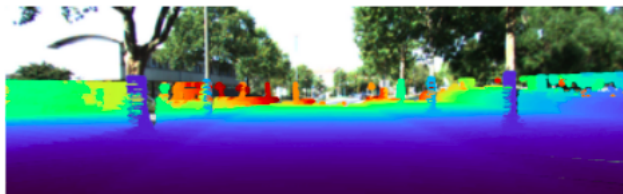


Figure 1.1: Example of sparse depth map provided by a LIDAR sensor. Note the lack of information in the upper part of the image due to sensor limitations.

## 1.3 Image-based depth estimation

A cheaper and overall more robust alternative lies in recovering depth from standard camera images. The most straightforward but more expensive solution relies on two cameras whose image planes are coplanar and are displaced horizontally

similar to our eyes. The horizontal baseline between the two cameras creates a parallax that causes a scene point to be projected to different positions on the respective images. The displacement between the projection positions on the two images is called disparity. As shown in Figure 1.2, if we know the cameras' calibration parameters and correspondences of pixels between the two images, we can measure disparity in pixels, convert it to meters and then recover the depth of the pixel as the inverse of the disparity. Theoretically, given that all pixel correspondences are known, this method can provide dense depth maps, meaning depth information for each pixel.

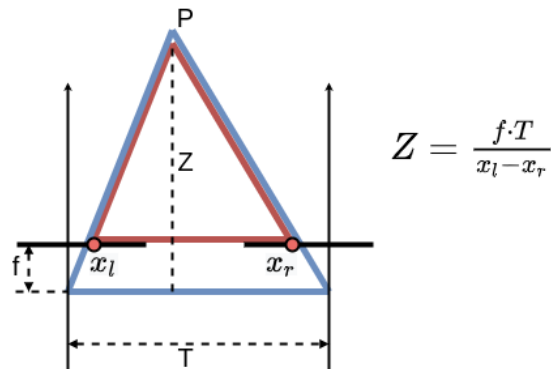


Figure 1.2: Given two cameras whose image planes are coplanar depth of point P can be recovered if the camera focal length and the position of point P in the two images are known.

### 1.3.1 Learning depth estimation

The correspondences can be estimated using traditional descriptor-based matching algorithms. These methods, however, do not allow to recover depth for uniformly colored objects and occluded regions. For this reason neural networks have been recently employed to learn the disparity map given two images captured from a stereo setup or even just a single image. This task is usually learned by a neu-

ral network using supervised training on datasets whose ground truth depth maps have been collected either using LIDARs or by generating the dataset in a simulator to provide dense pixel-perfect disparity maps.

## 1.4 Self-supervised training

Supervised training requires expensive sensors, such as LIDARs, to acquire accurate ground truth data. Moreover, the supervision offered by these sensors is sparse and often does not provide depth information for the upper part of the image due to limitations in the sensors, as shown in Figure 1.1. To overcome this issue self-supervision can be employed to learn depth by imposing geometrical constraints between the different camera images and their predicted depth maps.

This thesis aims at deployment of a depth estimation network on mobile phones that are not equipped with stereo camera setup. The self-supervised training scheme is therefore based on monocular video feeds. This allows fine-tuning of the network on a data distribution that is as close as possible to the target distribution at inference time. Unlike self-supervision from stereo images, this training scheme allows estimation of depth maps only up-to-scale. This is due to the lack of scale information in the pose between the camera frames used during training. Moreover, self-supervision from monocular videos often results in inaccuracies due to color inconsistencies between different images. To tackle the former problem, scale recovery using GPS velocity is employed. For the latter, sparse depth points provided by a monocular SLAM system is used to improve accuracy in harder regions of the images.

## **1.5 Why mobile phones as target platform**

Most recent works focus on estimating depth using an expensive GPU to achieve realtime performance with very complex networks and postprocessing steps.

In a real-world scenario, these devices cannot be employed due to money, power, and space limitations. Less expensive and more power-efficient devices such as the Jetson TX2 and Jetson Xavier improve on these limitations but still require dedicated mounting and a relatively high power budget, close to 10W. Mobile phones, on the other hand, offer a prebuilt platform, ready for deployment with integrated batteries and very low power requirements. They are easily mountable in many systems like cars that might benefit from depth-sensing capabilities. Moreover, many common mobile applications, such as bokeh simulation and augmented reality, rely on depth sensing.

## **1.6 Thesis Structure**

The first part of the thesis will cover the related work in Chapter 2 and an in-depth explanation of the Monodepth2 model by Godard et al. [5] in Chapter 3. In Chapter 4 all the techniques used to tackle the shortcomings of Monodepth2 will be shown. In 5 all the results will be reported. Finally, in Chapter 6 future possible developments will be discussed followed by the conclusions.

# Chapter 2

## Related Work

Previous works in monocular depth estimation can be split into supervised and self-supervised approaches. The former makes use of ground truth depth maps at training time the latter relies on image reconstruction. Both training scheme will be covered in the literature review, respectively in Section 2.1 and Section 2.2. Moreover, since the goal of the thesis is to run in real-time on mobile phones, papers about compact architectures and neural network optimization will also be discussed in a separate section. Finally, as a SLAM system will be used to generate sparse depth maps to improve the accuracy of the predictions, so the literature regarding this technology will also be covered.

### 2.1 Supervised Monocular Depth Estimation

Eigen et al. [6] were among the first to achieve high accuracy using convolutional layers for monocular depth estimation. They employed two networks to compute

depth at different scales and used supervised training with projections of LIDAR points as ground truth training samples. To tackle the problem of scale ambiguity, they introduced a relative depth error metric. Laina et al. [16] improved upon this by applying a fully convolutional neural network to the task, proposing the use of up-convolutions instead of the more common, but computationally expensive, deconvolutions. They also introduced the reverse Huber loss (BerHu) to better deal with the wide value range of depth. It allows to give relatively high importance to big errors, similar to an L2 loss but without disregarding small errors, similar to an L1 loss. All these works presented the depth estimation problem as a regression task. Fu et al. [7] took a different path and cast the depth estimation problem as an ordinal regression task, asking the network to predict, for each pixel, a probability distribution over a set of possible depths. Even though the depth maps produced suffered from clear discretization artifacts, this approach brought noticeable quantitative improvements, beating previous state-of-the-art methods by a wide margin. These limitations have been tackled by a number of subsequent works. In particular, Bhat et al. [4] achieved much better qualitative and quantitative results using an attention module to correct the predictions, as well as predicting the bin widths as a function of the input image rather than using a predetermined static discretization. Raftl et al. [24] improved the robustness of monocular depth estimation by mixing different datasets during training. They proposed a robust training objective invariant to changes in depth range and scale. With their approach, they achieved high accuracy in zero-shot cross-dataset transfer tasks, notoriously difficult for monocular depth estimation.

## 2.2 Self-Supervised Monocular Depth Estimation

Depth supervision requires expensive hardware (usually LIDAR sensors) and often requires manual refinement of the ground truth to tackle hardware limitations. An alternative is given by self-supervised depth estimation, where given two images taken from different points of view, one is reconstructed from the other through the prediction of the disparity for each pixel. The warping operation used to reconstruct an image from the other is differentiable thanks to the development of spatial transformer networks (STN) [15]. Godard et al. [10] used STNs and stereo images for self-supervised training. They found that predicting both the left and right disparity map and introducing a left-right consistency loss helps to stabilize the training and learn accurate dense disparity maps. Their network was able to beat even the supervised approaches available at the time. Successive work tried to improve the robustness and accuracy of the predictions, by employing different, novel loss functions and more complex networks. Aleotti et al. [1] improved upon the commonly used appearance loss by implementing a discriminator whose task was to discern ground truth images from the reconstructed ones. Thanks to this adversarial approach, they measured a slight accuracy improvement compared to Godard.

Another strong assumption behind the photometric loss is the presence in the scene of only lambertian materials, meaning that objects appear uniformly bright from all directions of view. Since this assumption is often violated, Shu et al. [26] proposed to add a feature-based loss to the standard photometric loss. They use an autoencoder to learn a feature representation for the images and then compute the loss for monocular depth estimation on the features. The auto-encoder is regularized based on desired properties for the features, namely smoothness and magnitude of gradient. Andraghetti et al. [2] achieved better metrics by adding as

input sparse depths computed with a traditional visual odometry algorithm. These were densified using a sparse autoencoder based on sparse invariant convolutions [30] and then used as additional input to the depth estimation network. Huynh et al. [14] employed feature alignment kernels to merge sparse points from SLAM to the feature maps extracted from the images. In Bello et al. [11] the same ordinal regression approach of [7] is used in a self-supervised framework. Using the predicted disparity probability volumes they compute an occlusion mask to ignore occluded regions during training. Moreover, they apply the same ordinal regression approach on previous networks, like MonoDepth [10], showing how discretized disparity bins generalize to other architectures too. Regarding self-supervised networks using monocular data for training, one of the biggest milestones is represented by Monodepth2 [5]. Since the pose between the different camera frames is unknown in this setting, they employed a PoseNet to predict it. By computing a mask for stationary pixels, they also took into account the problem of objects moving at the same speed as the camera, that would otherwise be estimated to be at infinite depth. The problem of scale ambiguity, a natural consequence of monocular training, was not addressed so at test time the predicted depth maps were scaled by comparing their median with the ground truth median value. This limitation was tackled in [25] where Guizilini et al. introduced a new network based on 3D convolutions and a new scale recovery method based on the velocity ground truth data acquired by the OXTS of the KITTI dataset [8].

## **2.3 Compact architectures for real-time performance**

The previously discussed works attempted to improve accuracy using complex networks. This leads to higher inference time, oftentimes struggling to achieve

real-time execution even on a high-end desktop GPU. However, in many real-world applications, real-time performance is desirable and computing resources available are limited. Many compact architectures have been proposed in the past, such as the MobileNet family [12]. However, they have not been widely tested on monocular depth estimation tasks, limiting the architecture investigation to the image classification task. As an example, squeeze and excitation attention modules [13] implemented attention mechanism in a lightweight fashion for convolutional neural networks. They have been shown to increase accuracy with minimal performance loss in a wide variety of networks [23]. More recently, an even better alternative has been proposed by Bello et al. [3] where the computation of feature maps has been discarded in favor of more computationally efficient lambda functions.

Regarding efficient architectures specifically built for depth estimation, Poggi et al. [21] proposed Pynet, a pyramidal encoder-decoder architecture to solve monocular depth estimation with acceptable accuracy and inference time even on low power devices like Raspberry Pi 3. FastDepth [35], uses depthwise separable convolution and nearest neighbor upsampling to optimize the decoder stage. Together with an encoder based on MobileNetV1 they achieve real-time performance on a Jetson TX2. Using optimized compilation and pruning (through NetAdapt [22]), they measure a further 4-fold response time improvement. In MiniNet, Liu et al. [17] change the encoder with a recurrent neural network to further lower the number of network weights and inference time. Wang [32] used RegNetY-06 [23] together a custom decoder block and thanks to wide ablation studies they achieved higher accuracy than FastDepth while maintaining real-time performance on Android using an ARM A76 CPU.

## 2.4 SLAM and Visual Odometry

Self-supervised depth estimation, particularly when tackled as a regression problem rather than ordinal regression, has the major pitfall of having many local minima [34] and struggling with noisy textured objects like trees and thin structures like poles. Traditional SLAM algorithms are instead tuned to exploit these image irregularities and often are capable of providing very accurate estimates but for only a few points in the image, making them a good choice to improve the shortcomings of monocular depth estimation networks [2]. ORB-SLAM [19] is one of the most used methods for monocular SLAM. To perform monocular SLAM and retrieve scale, additional assumptions, like camera height from the ground, or sensors are needed. In [20] ORB-SLAM was improved by allowing the retrieval of the scale factor using data provided by an IMU. On the other hand, Tian et al. [29] tried to tackle the problem using the assumption of fixed height from the ground. This assumption is often correct for driving datasets since the camera is fixed on the car.

There have also been attempts at using neural networks to retrieve the scale and improve SLAM and VO outputs. In CNN-SLAM [28] CNN are used for DenseSLAM with accurate scale recovery. More recently, Yang et al. [36] use neural networks to estimate pose, depth, and depth uncertainty in a fully self-supervised framework by adopting some innovative losses that take into account photometric inconsistencies between frames.

# Chapter 3

## The Baseline

In [5] Godard et al. presented Monodepth2, a network for monocular depth estimation. Their proposed self-supervised training scheme accommodates training data obtained from both stereo and monocular cameras. In the latter case, the network learns to estimate depth only up to a scale factor. Their method requires only camera images and camera calibration parameters during training, forgoing the expensive and sparse supervision of ground truth depth from LIDAR sensors. Monodepth2 is the primary baseline that sets the stage for the extensions experimented in this thesis. In this chapter, the Monodepth2 framework will be described highlighting the differences in the reimplementation compared to the original formulation described in the paper. The extensions and improvements experimented during the thesis will be described in Chapter 4.

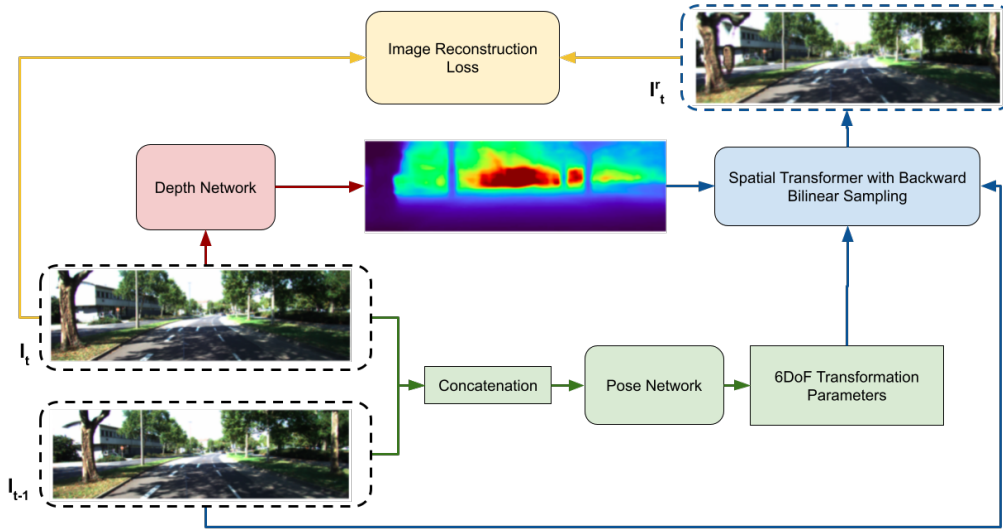


Figure 3.1: Scheme of Monodepth2 pipeline. This is a simplification with only two input frames, in the original paper and re-implementation done for this thesis 3 frames ( $t-1$ ,  $t$ ,  $t+1$ ) were used.

### 3.1 Depth estimation as Image Reconstruction

The core idea behind Monodepth2 is to cast the depth estimation problem as an image reconstruction task. The equation

$${}^i z' p' = p(Rp^{-1}(z, p) + t)$$

where

$$p(zp) = Kzp$$

$$p^{-1}(z, p) = K^{-1}zp$$

describes the relationship between two images of the same static scene taken from

two different points of views.  $K$  is the camera intrinsic matrix,  $z$  and  $p$  are the original depth and pixel homogeneous coordinates with 1 as last element,  $z'$  and  $p'$  are the depth and pixel coordinates after the transformation, from first camera to the second, represented by the rotation matrix  $R$  and translation vector  $T$ . Using this relationship it is possible to reconstruct an image in a differentiable way using spatial transformer networks [15] with backward bilinear sampling. It should be noted that this formula is valid only under constant brightness and Lambertian surface assumptions. These are violated in realistic scenes and apposite techniques to tackle them will be presented in Section 3.3.

When training using a pair of stereo images, the relative camera pose is usually static and known beforehand, just like the intrinsic matrix. In this setup, only a depth network is needed to predict depth that will be used to warp the source image into the target image reference frame.

When training using monocular videos, on the other hand, the relative camera pose is unknown, and constantly changing during the video. The camera pose between two frames can be estimated in a number of ways. We experimented both with a traditional monocular SLAM algorithm and with an apposite neural network to predict the pose alongside the depth. The latter is the approach used in Monodepth2 as shown in Figure 3.1.

## 3.2 Network Structure

The depth network used by Godard et al. is an encoder-decoder model with skip connections, with the structure shown in Figure 3.2. Each one of the multiscale outputs was passed through a sigmoid activation function and used to generate a

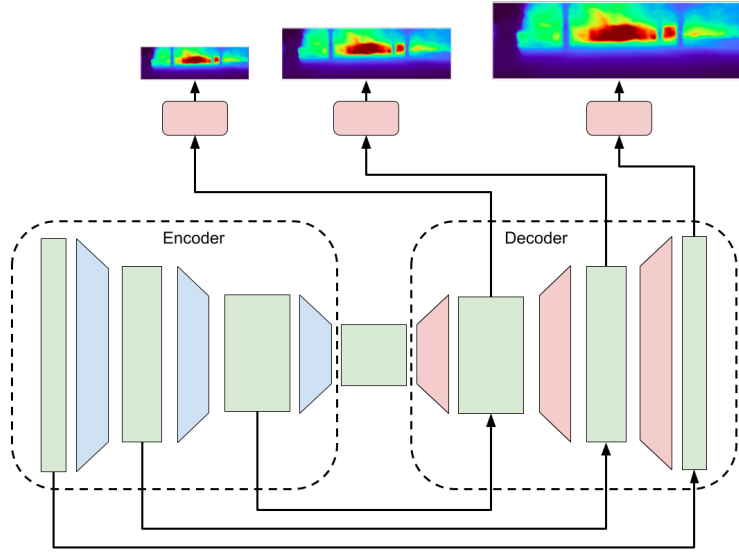


Figure 3.2: Depth network used by Godard et al. It is composed by an encoder a decoder and multiple disparity heads that compute the disparity maps, later converted to depth maps, at multiple scales.

scaled disparity map with:

$$D = d_{min} + (d_{max} - d_{min}) * D_{sig}$$

where  $D_{sig}$  is the output of the network after the sigmoid activation function and  $D$  is the scaled disparity map.  $d_{min}$  and  $d_{max}$  are the minimum and maximum disparity that the network can predict.

The encoder model in Monodepth2 uses a ResNet18 while the decoder upsamples feature maps via nearest neighbor upsampling followed by 2D convolutions. It is important to mention that Godard et al. measured better results when using reflective padding instead of zero padding.

When training using sequence data and not only stereo data, an additional network, the pose network was used to predict the 6 degrees of freedom (6DoF) transformations between the camera frames of the input images. This network

is a separate ResNet18 whose input is the concatenation, along the image channel axis, of all  $n$  input frames. The output, computed by a fully-connected layer, are all the  $6n$  parameters of the relative camera poses:  $3n$  rotation parameters in axis-angle representation and  $3n$  translations.

### 3.3 Losses

The training loss function employed by Godard et al. is a weighted combination of two main components, namely a masked photometric term  $L_p$  and a smoothness term  $L_s$ , giving a total loss of

$$L = mL_p + lL_s$$

Since the L1 error is not robust to brightness changes, it's not suitable by itself as photometric loss. For this reason the photometric error  $pe(I_a, I_b)$  between image  $I_a$  and image  $I_b$  is computed as:

$$pe(I_a, I_b) = \frac{a}{2}(1 - SSIM(I_a, I_b)) + (1 - a)\|I_a - I_b\|_1$$

where SSIM is the structural similarity index as described in [33].

To improve the disparity map in occluded regions, only the minimum photometric error between all the reconstructed images  $I_{t' \rightarrow t}$  and the target image  $I_t$  is used during training. Therefore,

$$L_p = \min_{t'} pe(I_{t' \rightarrow t}, I_t)$$

Moving objects are another major problem during the training. In particular, the position of the objects moving at the same speed as the camera, as projected onto the image plane, will not change between frames when training using only monocular videos. For example, this scenario could arise on highways with cars in front of and with the same speed as the camera. This will lead the network to learn an infinite depth for such objects. To tackle this problem, the weight term  $m$  is set to zero in all pixels where the photometric error between the source images  $I_{t'}$  and the target image  $I_t$  is lower than the one with the reconstructed images  $I_{t' \rightarrow t}$ .

$$m = [\min_{t'} pe(I_{t' \rightarrow t}, I_t) > \min_{t'} pe(I_{t'}, I_t)]$$

This allows the network to ignore all the regions that do not change over time, avoiding the pitfall of predicting infinite depth for them. The smoothness loss  $L_S$  regularizes the predictions based on the assumption that depth is locally smooth. To account for object edges, an edge-aware term is added based on the gradient of the image. Following [11] we add an additional coefficient  $l$  to choose how strongly the edge aware component should impact the loss.

$$L_S = |\nabla_x D_t| e^{-l |\nabla_x I_t|} + |\nabla_y D_t| e^{-l |\nabla_y I_t|}$$

## 3.4 Experimental Setup

### 3.4.1 KITTI Dataset

The KITTI Dataset [9] provides a reliable benchmark for autonomous driving task, such as depth prediction, stereo matching, optical flow, visual odometry and object detection. The dataset was captured by driving in a vehicle equipped with

two grayscale and two color cameras with a baseline of 0.54 meters. Accurate ground truth depth data is provided by a velodyne laser scanner. Accurate poses are possible thanks to a GPS localization system with integrated inertial measurement unit. The datasets are captured by driving in mid-sized cities, rural areas and highways.

In its raw form, the dataset contains 42,382 frames from 61 scenes, with a typical image being 1242×375 pixels in size. We employed two different splits for the data. The first one, referred from here on as Eigen Split [6], contains 22,600 frames in the train set and 697 frames in the test set. The second one, from here on called Odometry Split, contains 13,200 frames in the train set and 8691 frames in the test set.

### **3.4.2 Stockholm Dataset**

The Stockholm dataset was captured using a Pixel 5 mounted on the windshield of a car. It provides 1280x720 RGB images captures at 30 Hz, camera matrix and GPS-based location and speed gathered once every second. To make scale training easier we apply linear interpolation to the GPS speed in order to have measurements for all frames. Ground truth LIDAR data is not available for this dataset, therefore accurate depth accuracy measurement is not possible. Nonetheless we can have an estimation for the accuracy by comparing the predicted depths with the depth from object size as described in Section 3.4.4.

The dataset is manually filtered in order to remove scenes where the car is static and the long sequences on highways that would lead to learning the wrong depth for cars moving at the same speed of the camera. After filtering, the dataset contains 43900 images for the train split and 1500 images for the test set.

### 3.4.3 Depth Metrics

Evaluation of depth is based on 6 metrics for test sets with ground truth depth maps. The evaluation is performed by resizing, with bilinear interpolation, the predicted disparity maps to the ground truth height and width. Unless otherwise noted, the evaluation for networks trained using stereo data (S), velocity data (Vel) or sparse depth maps data (SD) is carried out on the raw output of the network without additional median scaling to align the scale with the ground truth. Network trained using standard monocular training make use instead of depth map scaling at test time. In the following sections  $D_{gt}$  is the ground truth sparse depth map,  $D_{pred}$  the predicted dense depth map.  $V$  is the set of pixel coordinates for which  $D_{gt}$  contains a valid (non-zero) pixel.  $N$  is the cardinality of  $V$ . Evaluation is done only for pixels whose ground truth depth is known and the following metrics were the one used.

#### Absolute Relative Error

$$\frac{1}{N} \sum_{(i,j) \in N} \frac{|D_{gt}(i,j) - D_{pred}(i,j)|}{D_{gt}(i,j)}$$

#### Squared Relative Error

$$\frac{1}{N} \sum_{(i,j) \in N} \frac{||D_{gt}(i,j) - D_{pred}(i,j)||^2}{D_{gt}(i,j)}$$

#### Root Mean Squared Error

$$\sqrt{\frac{1}{N} \sum_{(i,j) \in N} |D_{gt}(i,j) - D_{pred}(i,j)|^2}$$

## Threshold

$$\frac{\#\{D_{gt}(i, j) : \max\{\frac{D_{pred}(i, j)}{D_{gt}(i, j)}, \frac{D_{gt}(i, j)}{D_{pred}(i, j)}\} < 1.25^k\}}{\#\{D_{gt}(i, j)\}} \quad (3.1)$$

where  $k$  is the level of the threshold and  $\#M$  denotes the cardinality of set  $M$ . We used  $k = 1, 2, 3$ .

### 3.4.4 Depth from object size

For the Stockholm dataset 5.2, we don't have any ground truth depth map available. In order to have a general idea of how accurate, at least in terms of scale, the depth prediction is, we extrapolate depth information from this datasets using projective geometry.

Since some objects size is usually known, we can recover the distance of these objects from the camera along the principal axis if we know their size in the real world, in the image and the camera matrix. This method to recover depth from object size is shown in Figure 3.3.

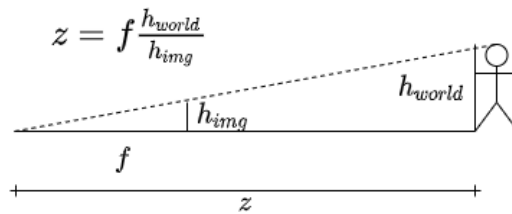


Figure 3.3: How to recover depth of an object, given his real world and image sizes.

Traffic signs are the perfect candidate for this task since their size is usually standardized in a country. Moreover, thanks to their mounting location, their height isn't usually distorted due to camera angles so recovering depth from their size would be quite robust. One downside of using traffic sign is the necessity to label

all the images in the dataset manually since commonly pretrained object detectors don't provide accurate enough bounding boxes for traffic signs far away. On top of this, the predicted disparity maps struggle particularly in representing traffic signs so there would be a lot of noise in the predictions even with perfect depth information from traffic signs size.

For these reason we decided to use car size to measure depth. Compared to traffic signs, cars are seen from a wide variety of angles and their height in the image gets therefore distorted more. Moreover their height in the real world is not the same for all cars, albeit there isn't a huge variation for most of them. But most importantly, even when further away from the camera, currently available object detectors like YOLO-V3 are capable of detecting quite accurate bounding boxes. And finally, since cars are bigger than traffic signs, the depth network usually doesn't completely miss their detection in the depth map.

Once chosen how to get a depth estimation from objects size in the image, we need to choose what depth from the prediction of the network to assign to this object. Since the bounding box can cover a wide area of the image, a wide range of depths might fall in the box. For this reason taking the average of the depths in the bounding box doesn't provide a good estimation of the predicted depth for the car (Figure 3.4). We investigated two alternatives: taking the median of the depths and the minimum. The former is motivated by the fact that in most cases the object covers the biggest portion of the bounding box so the median of the predicted depths would very likely fall into the object region. The latter was tested because object height is usually dependant on the closest part of the object, since it appears bigger.

Figure 3.4 shows the scatter plot of the depth predicted based on car height and the median and minimum approach applied to the ground truth depth map. As can be seen, the best alignment between the ground truth and depth based on object

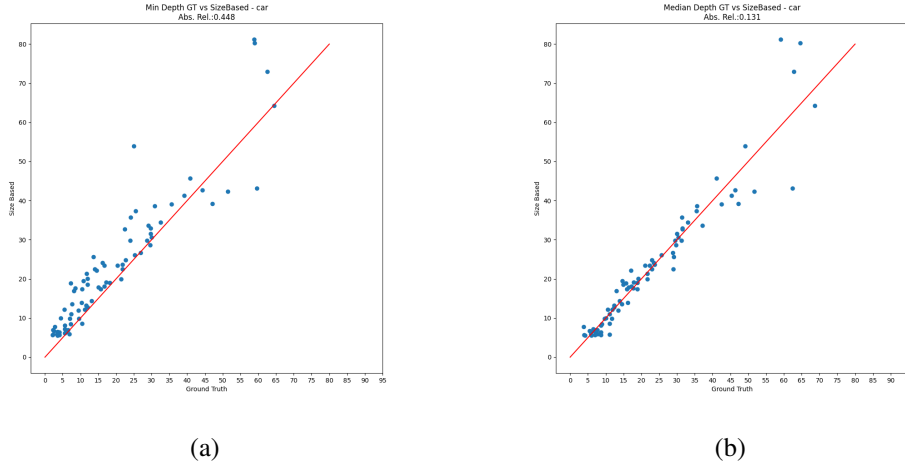


Figure 3.4: Scatter plot comparing how well the estimated depth using bounding box height aligns with the minimum (a) and median (b) values of the ground truth depth maps in the bounding box.

size is obtained when taking the median of the ground truth depths contained in the bounding box. 3.3.

### 3.5 Pose Metrics

Since we also have a pose network that predicts relative pose between two frames, we decided to measure how accurate the pose predicted by the network was. This test is mostly for informative purposes, since the focus wasn't on improving the pose prediction but just improve the depth map accuracy.

To evaluate pose we use 2 metrics following the practice for the KITTI odometry benchmark [8]. We evaluate the errors for a set of sequence lengths, from 100m to 800m with 100m steps. We then average them out to obtain our final error metrics. Given two frames  $i$  and  $j$ ,  $P_{gt}(i, j)$  and  $P_{pred}(i, j)$  are the ground truth and predicted relative camera pose transformation matrix between frame  $i$  and

$j$ .  $P_{err}(i, j) = P_{pred}(i, j)P_{gt}^{-1}(i, j)$  is the error between the two poses. We can then convert  $P_{err}(i, j)$  in axis-angle vector  $R_{err}(i, j)$  for the rotation and translation vector  $T_{err}(i, j)$ . Defining  $V$  the set of frames for which we have ground truth poses and  $L$  the set of sequence lengths we evaluate, we get the following error metrics:

#### Average translation error

$$\frac{1}{\#L\#V} \mathring{a} \mathring{a} \frac{\|T_{err}(i, i+l)\|}{\|T_{gt}(i, i+l)\|}$$

#### Average rotation error

$$\frac{1}{\#L\#V} \mathring{a} \mathring{a} \frac{\|R_{err}(i, i+l)\|}{\|T_{gt}(i, i+l)\|}$$

### 3.6 Implementation Details

Unless otherwise specified, all the experiments were conducted with the following parameters. Some of the parameters are slightly different compared to Monodepth2 mainly for computational reasons with our ablation study confirming the negligible change in accuracy. As an example, when training using stereo data the baseline was set to 0.54 compared to 0.1 of Monodepth2. This allowed removing the post-scaling factor that was used in the official implementation to correct the scale of the predicted depth. Experiments confirmed that the accuracy remained unchanged.

$d_{min}$  and  $d_{max}$  were set respectively to 1m and 200m, allowing to cover the full range of the KITTI dataset ground truth depths (2m - 80m) while providing some

margin for different datasets with different camera angles.

Since the networks were trained on an NVidia RTX 3090, compared to the Titan Xp used for the original paper, it was possible to increase the batch size from the 12 used by Godard et al. to 16. The input image resolution was set to 640x192 pixels.

The optimizer employed was Adam with starting learning rate of  $10^{-4}$ ,  $b_1 = 0.9$ ,  $b_2 = 0.999$  and  $e = 10^{-7}$ .

Unless otherwise noted, all networks are trained for 30 epochs dividing the learning rate by 3 at epoch 20 and again at epoch 25. Data augmentation was performed on the fly with 50% chance of horizontal flip and with the following color augmentations: random brightness ( $\pm 20\%$ ), contrast ( $\pm 20\%$ ), saturation ( $\pm 20\%$ ) and hue ( $\pm 10\%$ ).

Following the approach of Godard et al. [5], the color-augmented images were given as input to the network but, for the loss computation, non-color-augmented versions were used.

A noticeable difference compared to the original paper lies in the computation of the loss only at the highest output scale rather than all the four scales, as done by the original paper. Moreover, the loss was computed by working at the output resolution rather than up-sampling the disparity map at the original input image resolution. Regarding the inference time evaluation, all experiments were run on a Redmi Note 10 5G. Unless otherwise noted, all models were run in floating point inference mode using the Tensorflow Lite GPU Delegate with the precision loss flag enabled. The inference time was always measured by running inference at 640x352 input resolution. This was chosen as the closest resolution to the original phone camera aspect ratio that could also be executed on all networks employed, most of which apply stride 32 so they require an input whose resolution is a multiple of 32.

# Chapter 4

## Experimentation

The baseline by Godard et al. uses a relatively complex network that is not suitable for real-time inference on mobile phones. Moreover, there are some severe limitations regarding correctly scaled disparity maps when training with only monocular videos. It also did not cover a real-world deployment scenario where camera intrinsic and mount position could be very different compared to the training set. As shown in [31] these changes can greatly affect the accuracy of the predictions.

A wide range of experiments has been run to find a more efficient architecture (Section 4.1), improve accuracy with SLAM sparse depth maps (Section 4.2) and recover scale while training using only monocular data (Section 4.3).

### 4.1 Faster Neural Networks

The experiments regarding different neural networks for depth and pose estimation can be subdivided into three main sections. Regarding the depth network, the

encoder-decoder structure with skip connection was kept constant while the architectures of the encoder and decoder were changed to find a good balance between inference speed and accuracy.

The pose network, on the other hand, is not needed at inference time. For this reason it was not under any inference time requirements and the only reason to experiment with different pose network architectures was to achieve better depth accuracy.

### 4.1.1 Encoders

The MobileNet family has historically performed well in computer vision tasks when under low inference time constraints on mobile platforms like smartphones. For this reason, both MobileNet 2 and MobileNet 3 [12] were tried as encoders. As shown in Figure 4.1, both of these networks use inverted bottleneck blocks. The information passed between each block has a low number of channels that is then expanded by 1x1 convolutions before being processed with 3 by 3 depthwise convolutions. MobileNet 3 introduces also a squeeze and excitation module to the block. This raises the question of whether to apply the skip connection before

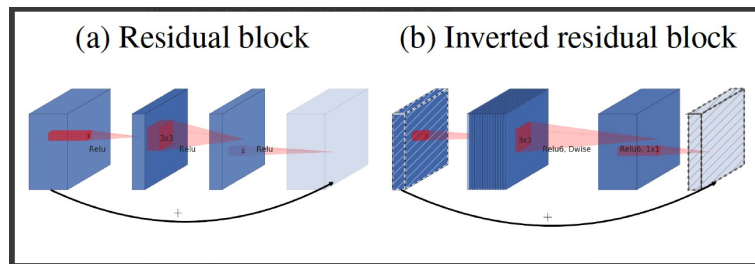


Figure 4.1: Inverted residual blocks used in the MobileNet architecture. In the original residual block, the number of channels is compressed with 1 by 1 convolution before applying a 3 by 3 convolution. In inverted residual blocks, the number of channels is instead lowered between blocks and expanded before applying a depthwise convolution.

or after the expansion. The experiments in Table 4.1 show that skip connections should be applied before expansion for MobileNets.

KITTI-Odometry - Eigen Crop		LOWER IS BETTER				HIGHER IS BETTER		
		Inference Time	Abs Rel	Sq Rel	RMSE	$d < 1.25$	$d < 1.25^2$	$d < 1.25^3$
EfficientNetB0	Before Expansion	130 ms	0.142	0.788	4.431	0.813	0.944	0.979
	After Expansion	179 ms	0.139	0.753	4.360	0.820	0.946	0.980
MobileNetV3	Before Expansion	83 ms	0.139	0.759	4.353	0.820	0.947	0.980
	After Expansion	138 ms	0.142	0.787	4.400	0.818	0.944	0.979

Table 4.1: Results on KITTI Odometry sequences 0, 4, 5, 7 using the Eigen Crop. Depending on the encoder architecture it might be more beneficial to apply the skip connections before or after the expansion in the residual block. Nonetheless, the accuracy gain doesn't justify the inference time increase.

Another architecture experimented with was PydNet [21]. This architecture was specifically designed for depth estimation at low latency on embedded hardware. As can be seen in Figure 4.2 the encoder of PydNet is a very simple stack of convolutional layers and most of the complexity of the network is in the decoder blocks. Using a different decoder we were able to improve the inference time as described in Section 4.1.2. Thanks to the newfound inference time margin, an additional variation of the PydNet encoder has been tried, with an higher number of channels in the early stages of the network.

As final comparison, EfficientNet [27] usually outperforms MobileNet in image classification tasks while having a smaller amount of parameters. Unfortunately, when it comes to inference on mobile phones, the EfficientNet models are measurably slower than the MobileNet equivalents. This trade-off is justified if we measured for depth estimation the same accuracy gains the architecture offers for image classification. As can be seen in Table 4.2, this was not measured in our experiments, where EfficientNet network performed comparably to MobileNet while being much slower. In the end, we found that PydNet with an expanded number of channels for the early stages gave the best trade-off between inference time and accuracy, providing the highest accuracy among the networks we exper-

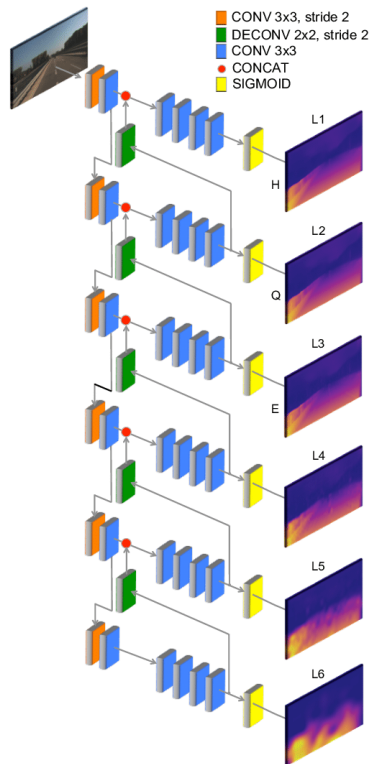


Figure 4.2: PydNet architecture. Note the use of very slow deconvolution layers instead of faster but less accurate nearest neighbor upsampling.

mented with, while retaining satisfying inference time at deployment. We will therefore, otherwise explicitly noted, use this type of encoder in all the following experiments.

When aiming at even faster inference times, the PydNet encoder with a lower number of channels or MobileNetV3 with skip connections before expansion proves to be a good choice if the accuracy loss is acceptable for the task.

The phone used during the thesis is slower than the target for the final deployment. On average, the Redmi Note 10 has double the inference time. Therefore, we chose to target a maximum inference time of 180 ms on the Redmi Note 10, allowing realtime inference on the deployment platform with some margin for overhead.

KITTI-Odometry - Eigen crop	LOWER IS BETTER				HIGHER IS BETTER		
	Inference Time	Abs Rel	Sq Rel	RMSE	$d < 1.25$	$d < 1.25^2$	$d < 1.25^3$
Pydnet16	47 ms	0.144	0.798	4.405	0.810	0.941	0.978
Pydnet32	77 ms	0.139	0.746	4.279	0.822	0.946	0.980
Pydnet64	141 ms	0.136	0.720	4.225	0.829	0.948	0.980
MobileNetV3 Large - After Expansion	138 ms	0.142	0.787	4.400	0.818	0.944	0.979
MobileNetV3 Large - Before Expansion	83 ms	0.139	0.759	4.353	0.820	0.947	0.980
MobileNetV3 Small - After Expansion	76 ms	0.147	0.840	4.578	0.802	0.940	0.978
MobileNetV2 - After Expansion	139 ms	0.138	0.742	4.279	0.823	0.946	0.980
EfficientNetB0 - Before Expansion	130 ms	0.142	0.788	4.431	0.813	0.944	0.979
EfficientNetB0 - After Expansion	179 ms	0.139	0.753	4.360	0.820	0.946	0.980

Table 4.2: Results on KITTI odometry sequences 0, 4, 5, 7. For different encoders. Inference Time was measured using the Tensorflow Lite GPU Delegate on a Redmi Note 10 5G at 640x352 input resolution.

## 4.1.2 Decoders

As previously mentioned, the architecture used also allows experimentation with different decoders. In the original Monodepth2 architecture, the decoder made use of mirror padding, with exponential linear unit activations and nearest neighbor upsampling as shown in Figure 4.3. While being a relatively efficient decoder, thanks to the use of nearest neighbor upsampling rather than deconvolutions, it still has a very computationally expensive structure since it was not designed for real-time inference on embedded hardware.

We also experimented with the baseline as well as the best decoder architecture found by Wang [32]. The former is a simple decoder block using 5 by 5 depthwise convolutions (Figure 4.4). The latter (Figure 4.5) inherits the idea of splitting the number of channels from the split-shuffle architecture [18]. This decreases the computational complexity of the convolutional layers and to allow communication between branches it shuffles the channels between each block. Since optimized implementations of the split and shuffle operations were not present in the framework used to deploy on mobile phones, the block was modified by removing the split and shuffle operation and adding a depthwise convolution in the second

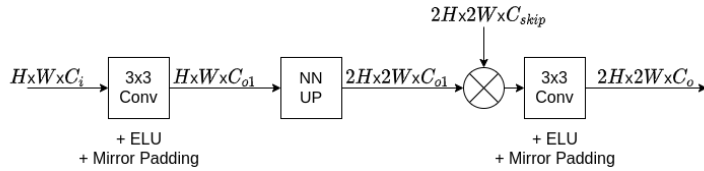


Figure 4.3: Monodepth2 Decoder Block

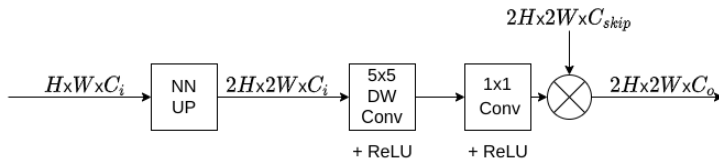


Figure 4.4: Naive depthwise convolution decoder block.

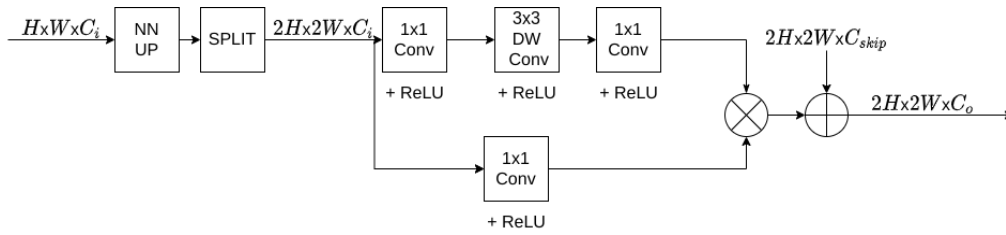


Figure 4.5: Decoder block from MobileDepth [32] inspired by the SplitShuffle architecture. Each block splits the work into two branches, applying 3 by 3 convolutions only in one of them. A shuffling of the channels at the end of the block allows communication between the two branches.

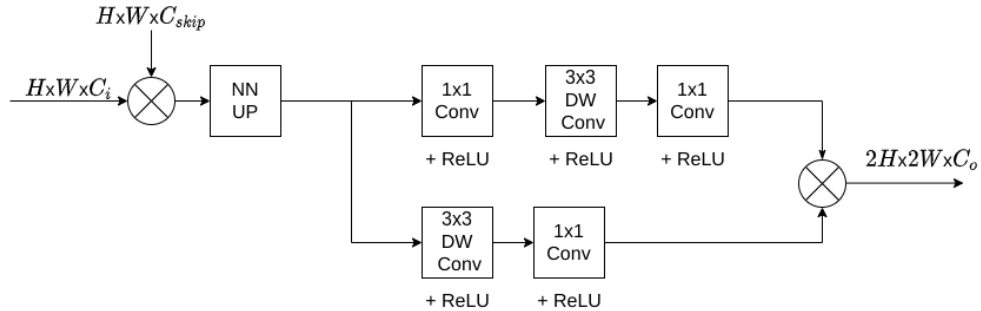


Figure 4.6: Modification of the SplitShuffle decoder block. It trades inference times for higher accuracy. The computational complexity is still low thanks to the halved number of channels.

branch too, following more closely the structure of the ShuffleNet 2 [18] block when the spatial resolution changes.

From the ablation study on the decoders, shown in Table 4.3, it can be seen that the best decoder depends on the target inference time and the available hardware. The MobileDepth block runs very quickly but the accuracy is among the lowest. Our block is 25% slower but improves upon all metrics measurably. Finally, if the deployment hardware allows for the higher computational complexity, the Monodepth2 original decoder structure achieves the best results while running 57% slower than our proposed block. Interestingly, increasing the number of channels of our decoder block, while heavily influencing the inference time, does not lead to better accuracy.

KITTI-Odometry - Eigen crop	LOWER IS BETTER				HIGHER IS BETTER		
	Inference Time	Abs Rel	Sq Rel	RMSE	$d < 1.25$	$d < 1.25^2$	$d < 1.25^3$
MobileDepth SplitShuffle	113 ms	0.139	0.728	4.280	0.822	0.947	0.981
Simple Depthwise CNN Decoder	122 ms	0.140	0.736	4.335	0.817	0.946	0.981
Ours	141 ms	0.135	0.717	4.220	0.830	0.948	0.981
Pydnet Decoder	198 ms	0.134	0.694	4.151	0.834	0.951	0.981
Monodepth2 Decoder	220 ms	0.132	0.703	4.153	0.836	0.950	0.981
Ours - Higher number of channels	437 ms	0.136	0.724	4.211	0.830	0.949	0.980

Table 4.3: Results on KITTI odometry sequences 0, 4, 5, 7 for different decoders.

### 4.1.3 Pose Networks

Unlike training from stereo images, training from monocular videos requires the prediction of the pose between the different input images. The accuracy of this pose can influence greatly the accuracy of the depth predictions [5] since, if the predicted pose is wrong, the depth map that minimizes the image reconstruction loss will also be incorrect.

One key difference with the depth network is the lack of any inference time requirements on the pose estimation network. This is due to the fact that at deploy-

ment we infer depth only on one frame so we do not need to know the relative camera pose between the previous frames. The only drawback of overly complex pose networks is the slower training, and in some cases, the need to reduce the batch size due to GPU memory restrictions.

Monocular SLAM systems offer an alternative to training a pose network. We use the pose predicted by the SLAM system. The actual pose evaluation will be covered more in detail in Section 5.3.

Following Monodepth2, we predict pose with an axis-angle representation and 0.01 scale. This means that the head of our networks is a fully connected layer with  $6n$  output values to represent the  $n$  camera transformations between the  $n+1$  frames given as input. The output is multiplied by 0.01 to allow for better initialization.

Our baseline is the Monodepth2 pose network based on a ResNet18 backbone. As an alternative, we also used MobileNetV2, EfficientNetB0 and EfficientNetB4 to investigate the possibility of achieving similar or improved accuracy, while changing the complexity of the backbone. The network head remained unchanged with a global average pooling layer applied at the last output of the backbone followed by a fully-connected layer.

As can be seen in Table 4.4 our baseline performed best in both accuracy and training time. It is worth noting that the SLAM system albeit providing a more accurate pose estimation (Section 5.3), leads to a much worse depth accuracy. Finally, the more complex architecture of EfficientNetB4 does not manage to improve the accuracy of the estimated depths albeit training much slower than our baseline. For these reason in all the following experiments we used ResNet18 for our pose network.

KITTI-Odometry - Eigen crop	LOWER IS BETTER				HIGHER IS BETTER		
	Training Time	Abs Rel	Sq Rel	RMSE	$d < 1.25$	$d < 1.25^2$	$d < 1.25^3$
Monodepth2 (ResNet18)	8h 30min	0.135	0.717	4.220	0.830	0.948	0.981
Zenuity VO	8h 35min	0.160	1.030	5.209	0.779	0.931	0.974
EfficientNetB0	10h 50min	0.138	0.740	4.253	0.826	0.949	0.981
EfficientNetB4	15h 59min	0.137	0.724	4.221	0.826	0.948	0.981
MobileNetV2	9h 58min	0.137	0.729	4.260	0.826	0.949	0.981

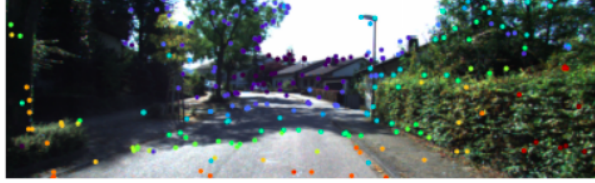
Table 4.4: Results on KITTI odometry sequences 0, 4, 5, 7 for different pose networks. The evaluation of the pose accuracy will be shown in Section 5.3

## 4.2 Sparse Depths

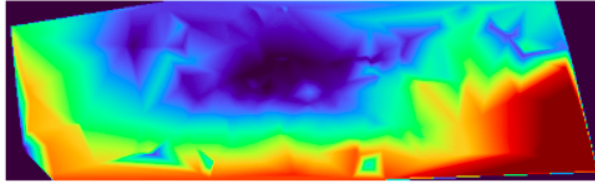
As previously mentioned, traditional SLAM algorithms provide often very accurate 3D points albeit very sparse in the image. Following previous work by Andraghetti et al. [2], the main goal of this section is to describe a possible improvement in accuracy thanks to the injection of 3D points computed by a monocular SLAM system into the pipeline.

The SLAM system used is not open source. It is chosen for the higher accuracy of its generated points than the likes of ORB-SLAM3. It provides also points at correct scale. On the other hand, the scale recovery method based on the IMU implemented in ORB-SLAM3 was not able to reliably recover the scale for many of the driving sequences used. This was probably due to the high noise in the IMU data and low acceleration values measured when the driving speed was constant. Just like ORB-SLAM3, the points generated by the SLAM algorithm used are very sparse in the image (Figure 4.7a) and this makes them unsuitable for standard convolutional neural networks that usually assume a dense input.

As shown in Table 4.6, concatenating the sparse depth maps with the color images results in a very small improvement. A much better alternative is provided by interpolating the sparse depth maps before concatenating them with the input image channels, as shown in Figure 4.7b. Both these options have the advantage



(a)



(b)

Figure 4.7: Sparse disparity maps generated by Zenuity monocular SLAM system (a) and interpolation of the sparse depth maps (b).

of being relatively inexpensive computationally.

The original work by Andraghetti et al. [2] ignored the computational complexity and achieved much better metrics by utilizing an autoencoder to interpolate the sparse depth maps. They made use of multiple sparsity invariant convolutional layers, originally developed by Uhrig et al. [30]. The re-implementation of the same approach showed that the accuracy improvements were replicated even with the different pipeline and networks used in this thesis. Our resulting network architecture was modified as shown in Figure 4.8.

Following [2] two additional losses were added to maximize the effectiveness of the new information available. The first, an inner loss  $L_{inner}$  to help the autoencoder train and enforce consistency between the input sparse depths and the output of the autoencoder.

$$L_{inner} = \frac{1}{N} \sum_{D_{sd}(i,j) > 0} |D_{dd}(i,j) - D_{sd}(i,j)|$$

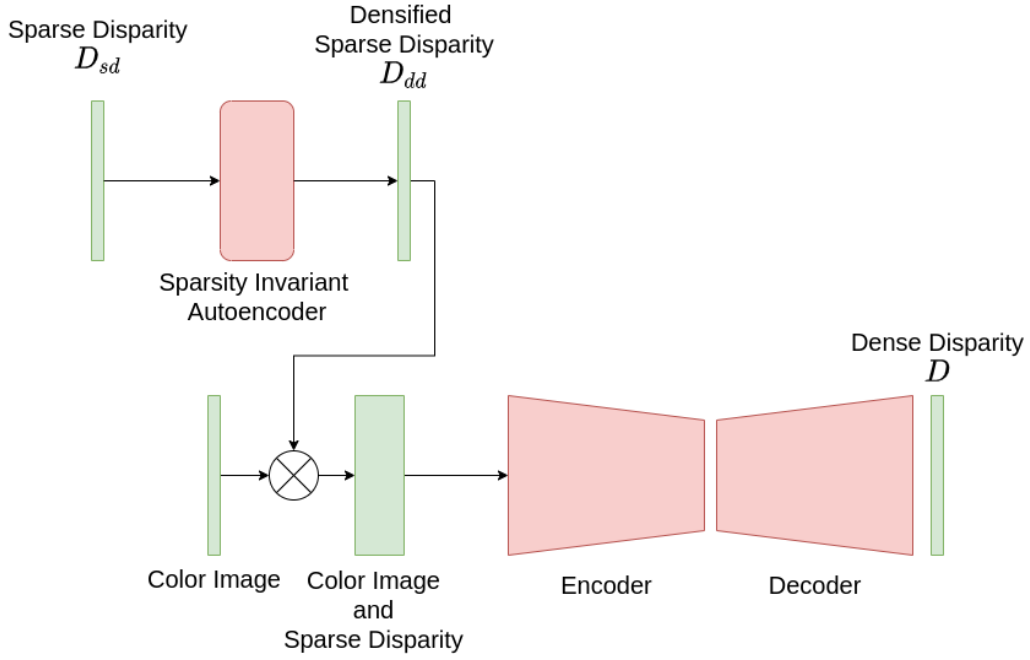


Figure 4.8: Network structure with sparse depths autoencoder.

The second loss added, the outer loss  $L_{outer}$ , was to enforce consistency between the input sparse depths and the predicted disparity maps.

$$L_{outer} = \frac{1}{N} \sum_{D_{sd}(i,j) > 0} |D(i,j) - D_{sd}(i,j)|$$

The total loss became as a consequence:

$$L = mL_p + lL_s + aL_{inner} + bL_{outer}$$

Ablation of the parameters  $a$  and  $b$  of these two losses can be seen in Table 4.5. These were run using the autoencoder structure originally developed in [2]. While in previous experiments the evaluation was run after scaling the predicted depth maps to have their median match with the median of the ground truth depth maps,

in this section we evaluate without median scaling to show what values for the outer loss do not allow scale recovery. The ablation study shows that an higher inner loss weight is slightly counterproductive. Regarding the outer loss weight, if too low it does not allow scale recovery, while with a too high  $b$  the accuracy drops again.

$a$	$b$	LOWER IS BETTER			HIGHER IS BETTER		
		Abs Rel	Sq Rel	RMSE	$d < 1.25$	$d < 1.25^2$	$d < 1.25^3$
0.01	0.01	0.488	3.862	9.425	0.010	0.040	0.618
0.1	0.01	0.458	3.468	9.031	0.013	0.067	0.789
0.5	0.01	0.503	4.076	9.739	0.009	0.031	0.425
0.01	0.1	0.104	0.706	3.962	0.869	0.947	0.975
0.1	0.1	0.105	0.728	3.961	0.868	0.947	0.975
0.5	0.1	0.105	0.734	3.990	0.867	0.945	0.974
0.01	0.5	0.110	0.848	4.115	0.863	0.943	0.972
0.1	0.5	0.111	0.832	4.048	0.864	0.944	0.973
0.5	0.5	0.110	0.813	4.034	0.865	0.945	0.973

Table 4.5: Results on KITTI odometry sequences 0, 4, 5, 7 for different values of alpha and beta when using the autoencoder from [2]. Median scaling with ground truth was disabled in this experiments.

Based on the assumption that the depth maps provided by the SLAM system are very sparse, it was observed that it was possible to lower the input and output resolution of the autoencoder while keeping most of the points. The lower resolution could allow us to run at a much higher inference speed while keeping most of the information. To do so, we applied max pooling with stride 4 to the input sparse disparity maps and then applied nearest neighbour upsampling to scale the output of the autoencoder up to the original resolution again. Finally, to further lower the inference time, we used a simpler autoencoder with fewer layers and smaller kernels.

As shown in Table 4.6, accuracy remained comparable but inference times were drastically reduced.

KITTI-Odometry Eigen crop	LOWER IS BETTER				HIGHER IS BETTER		
	Inference Time	Abs Rel	Sq Rel	RMSE	$d < 1.25$	$d < 1.25^2$	$d < 1.25^3$
No sparse depths	141 ms	0.134	0.7142	4.196	0.8323	0.9491	0.9807
Raw SD	143 ms	0.1155	0.6875	4.331	0.8412	0.9439	0.9759
Interpolated SD*	143 ms	<u>0.1051</u>	0.7146	<u>3.997</u>	<u>0.8656</u>	0.9463	0.9749
Autoencoder from [2]	406 ms	0.1056	0.7463	3.932	0.8721	0.9477	0.9747
Ours	152 ms	0.1035	0.6416	3.999	0.8649	<u>0.9482</u>	<u>0.9763</u>
Interpolated SD + Ours	152 ms	0.1068	<u>0.687</u>	4.062	0.8612	0.9476	0.976

Table 4.6: Results on KITTI odometry sequences 0, 4, 5, 7 for different methods to inject traditional SLAM system knowledge into monocular depth estimation. It should be noted that nor the SLAM algorithm nor the interpolation were included in the inference time measurements.

## 4.2.1 Masking Sparse depths

One shortcoming of the sparse depth points we used was the lack of occlusion handling during their generation. All points in the local map of the SLAM solution that were falling inside the camera field of view were projected in the sparse depth maps. This allowed more points per image but some very noticeable artefacts in scenes with moving objects (Figure 4.9). Since the code used to generate the point was not open source, an experiment was conducted to measure how much this type of error was impacting the results.



Figure 4.9: The sparse depth maps used were generated by using all the mapped points, falling inside the camera frustum, without taking into account possible occlusions. This leads to particularly relevant mistakes in the case of moving objects.

A pretrained object detection network was run on the training images to detect all the most common moving objects (cars, trucks, bicycles, pedestrians). The sparse depth maps were then filtered by removing all the points that fell into the predicted bounding boxes whose confidence score was above 0.3. The network

used was a Tensorflow 2 implementation of YOLO-V3 <sup>1</sup>, with weights pretrained on the COCO dataset.

In Table 4.7 we can see the comparison between the network trained with all the sparse points and the one trained with the masked sparse points using YOLO bounding boxes. Evaluation was performed both on the full frame and also only for the ground truth depth points falling inside the bounding boxes. The latter was run to measure how much the predictions improved on the masked regions that should be the ones to benefit more from this masking approach. A small improvement was measured for both evaluation regions, which confirms that the lack of occlusion handling is not a major bottleneck in the network’s depth maps accuracy.

KITTI-Odometry		LOWER IS BETTER			HIGHER IS BETTER		
		Abs Rel	Sq Rel	RMSE	$d < 1.25$	$d < 1.25^2$	$d < 1.25^3$
Eigen crop	Default	0.103	0.696	3.931	0.872	0.949	0.976
	Masked SD	0.099	0.631	3.854	0.876	0.952	0.977
Objects only	Default	0.198	2.729	6.488	0.735	0.861	0.921
	Masked SD	0.187	2.260	6.190	0.738	0.868	0.927

Table 4.7: Comparison between models using all the sparse depth map points and models not using points on cars. The Object only crop evaluates only on ground truth depth map points that are falling inside the vehicles’ bounding boxes.

### 4.3 Scale recovery

One of the major shortcomings of training a network using only monocular videos is the theoretical impossibility in recovering a correct scale for the predicted depth maps to estimate a per-image scaling factor. This leads most research on this field to correct the scale at test time by comparing the median of the ground truth with the median of the predicted depth maps. Since this is clearly not applicable to a

<sup>1</sup><https://github.com/schissmantics/yolo-tf2>

real world deployment, scale recovery is necessary either at train time or test time. The latter is not ideal since we want to keep computations at deployment as low as possible. For this reason we investigated scale recovery during training.

As seen in Section 4.2, training using an high enough coefficient for the outer loss allows the learning of correctly scaled depth maps. This works well but given the fact that the SLAM algorithm used is proprietary, additional investigations for a faster and cheaper alternative have been conducted.

One option could be to assume constant height from the ground and enforce such height by computing the ground plane from the predicted depth maps and then estimate the predicted camera height. One limitation of this approach is that it relies on constant height from the ground, that might not be always known if the phone is mounted in different vehicles. A more robust approach, proposed in [25], uses velocity information to enforce the correct scale at the pose network level. As a consequence, the predicted depth is learnt at a correct scale too. Velocity information can be acquired by GPS and, as long as there is no bias in the measurements, the network will learn the correct scale even when the speed measurements are noisy.

Following their approach, we added an additional loss term  $L_{vel}$ .

$$L_{vel}(t, t') = \left| \frac{\|\mathbf{v}_t + \mathbf{v}_{t'}\|}{2} Dt - \|\mathbf{t}_{t \rightarrow t'}\| \right|$$

Where  $\mathbf{t}_{t \rightarrow t'}$  is the translation predicted by the pose network between frame  $t$  and frame  $t'$ ,  $\mathbf{v}_t$  and  $\mathbf{v}_{t'}$  are respectively the ground truth velocity for the respective frames and  $Dt$  is the time difference between the two images timestamps. The final total loss for the network becomes:

$$L = mL_p + lL_s + aL_{inner} + bL_{outer} + gL_{vel}$$

Choosing the correct value for  $g$  is very important since if the value is too low the network will ignore the velocity loss component and not learn any scale. On the other hand, if the value is too high, the network will focus excessively on the scale and since the ground truth velocity has some noise it will hinder the learning of a stable pose leading to less stable training for the depth network too.

For these reasons, we conducted ablation studies on the  $g$  hyperparameter. In Table 4.8 the network was trained without the autoencoder, so without inner and outer loss too, to avoid interference with the scale learning. The best value found was  $g = 0.005$ . Higher values hindered slightly the accuracy of the depths maps. We assumed this is due to the added noise in the pose network gradients, making it less stable and harder to train the depth network. Lower values were not enough to get the network to learn depth.

$g$	LOWER IS BETTER			HIGHER IS BETTER		
	Abs Rel	Sq Rel	RMSE	$d < 1.25$	$d < 1.25^2$	$d < 1.25^3$
0.001	0.497	4.091	9.133	0.010	0.035	0.479
0.005	0.123	0.721	4.338	0.832	0.943	0.977
0.01	0.123	0.739	4.384	0.830	0.940	0.976
0.015	0.124	0.734	4.403	0.826	0.939	0.975
0.02	0.123	0.735	4.391	0.828	0.940	0.975

Table 4.8: Results on KITTI odometry sequences 0, 4, 5, 7 for different values of parameter  $g$  used to weight the velocity loss term.

## 4.4 Quantization

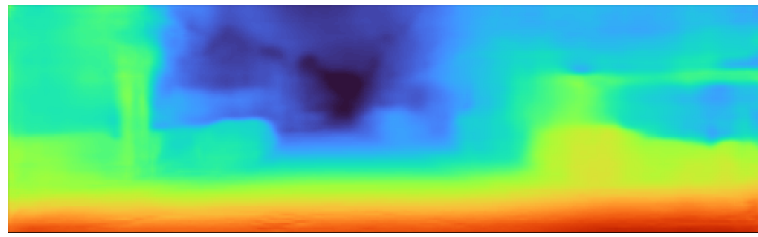
Quantization is a technique that allows to lower the FLOPs and memory required to run a model while retaining most of the accuracy. Moreover, it also allows to save space on the device even when inference cannot be speeded up. Given appropriate hardware and software frameworks, the computational complexity can be reduced too. As an example, integer operations are usually much faster and energy efficient than floating point arithmetic, therefore converting the model weights and

arithmetic to use only integers would be greatly beneficial.

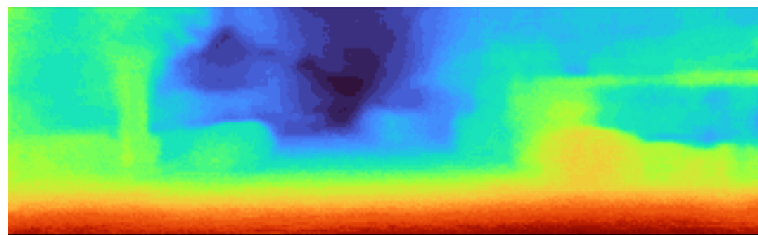
When using a lower number of bytes or numeric types to represent weights, not all software frameworks for deployment on mobile phones allow to experience the inference speed benefits, due to limitations in their implementations. These limitations are also related to the processor used. As an example, models whose weights have been quantized to 8-bits integers can run faster on mobile phones' CPUs but not on GPUs that resort to floating point 16-bit inference mode even for 8-bits integers models. This is a current limitation of the library we used and might change in the future library updates.

We therefore experiment with different quantizations methods and measure performance on all type of processor available on the Redmi Note 10 used for our experiments. As can be seen in Table 4.9, saving weights as 16-bits floats rather than 32-bits allows to double inference speed with an unnoticeable loss in accuracy on the test set. On the other hand quantizing our model to 8-bits integers allows higher inference speed on CPU but the accuracy drop is significantly higher. The generated depth maps are still usable although discretization artifacts are now much more noticeable, as can be seen in Figure 4.10. It should be noted that our phone hardware achieves lower inference time using 16-bits float on GPU rather than 8-bits integers on CPU. Nonetheless, since with the hardware and software library used, 8-bits quantization is the only way to noticeably speed-up computation on CPU, it might still be useful if inference on GPU cannot be run for technical reasons, such as overheating or other processes already using the GPU.

It is relevant to note that some mobile phones offer an additional processor to run neural networks on. For some of these, it can be accessed using the Hexagon Delegate of Tensorflow Lite, for others the NNAPI allows the exploitation of the neural network accelerator. The network model compatibility with the accelerator is fundamental. As an example, our network is currently not fully compatible with



(a)



(b)

Figure 4.10: Estimated depth map before (a) and after (b) 8bit integer quantization.

the Hexagon delegate and therefore doesn't benefit from faster inference times on phones that offer support for the delegate. On the other hand, MobileNet3 is fully compatible and the inference time is much lower when running fully 8-bit integer models on the accelerator (82ms vs 420ms). This shows how hardware and software really dictate the model choice. Unfortunately the Hexagon delegate is not compatible with many mobile phones therefore we didn't use it as a target. On the other hand GPUs are available on most mobile phones and with proper ventilation around the phone overheating can easily be prevented. GPUs are also the fastest processor for neural networks on many mobile phones. Therefore, all of the experiments in the thesis used 16-bits floating point arithmetic run on GPU.

Model	Quantization	INFERENCE TIME		METRICS	
		CPU	GPU	Abs Rel	$d < 1.25$
No Sparse Depths	Float32	645 ms	262 ms	0.123	0.832
No Sparse Depths	Float16	625 ms	141 ms	0.123	0.831
No Sparse Depths	Int8	541 ms	140 ms	0.128	0.809
Sparse Depths	Float32	732 ms	285 ms	0.104	0.865
Sparse Depths	Float16	722 ms	152 ms	0.104	0.865
Sparse Depths	Int8	612 ms	152 ms	0.108	0.860

Table 4.9: Results on KITTI odometry sequences 0, 4, 5, 7 for different quantization techniques.

## 4.5 Additional approaches tried

In the course of the thesis additional experiments were run that did not bring measurable improvements in the metrics but are worth mentioning.

### 4.5.1 Disparity Probability Volumes

In [7] Fu et al. proposed to cast depth estimation as an ordinal regression task. Therefore, the network has to predict a probability distribution over a set of possible depth bins. Bello et al. [11] extended the idea to self-supervision from stereo images. Following their proposal we experimented a similar approach on our networks both with stereo training and only monocular training.

Figure 4.11 shows the different training procedure used. The main idea is to predict a probability distribution over a list of  $N$  disparity bins for each pixel instead of a disparity map. The output of the network is the disparity logit volume  $D_L^L$ . We then warp multiple copies of the same image, one for each bin and the predicted disparity logit volume into the target image reference frame. We feed the warped disparity logit volume through a softmax function to generate the disparity probability volume  $D_L^{PR}$ . Finally, we generate our reconstructed image by weighting

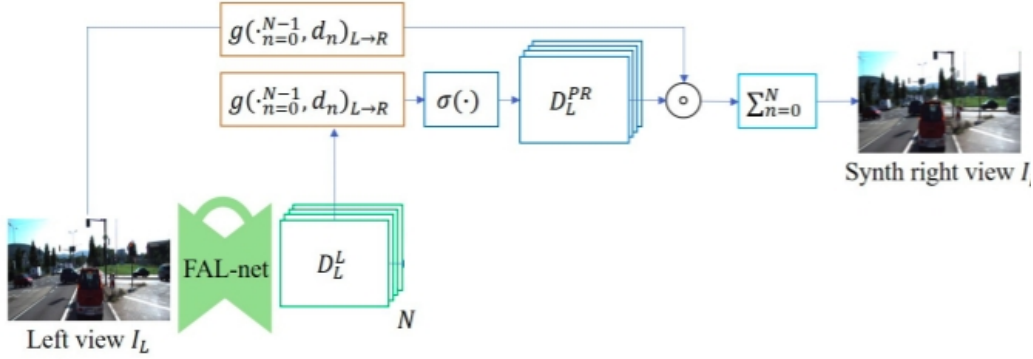


Figure 4.11: Disparity probability volumes pipeline for stereo training used by Bello et al. [11].

the differently warped images by the warped probability volumes for the given pixel and disparity bin. To be noted, instead of warping different images into the same camera reference frame, like for the standard regression self-supervision, we warp the disparity volume and the images into multiple cameras reference frame ( $t = +1, t = -1$ ) and then proceed with the rest of the pipeline as usual.

One major difference with the work by Bello et al. [11] is the lack of a left-right consistency loss and the lack of crop augmentation in our experiments. This was done to limit memory consumption and to keep consistency with the pipeline of previous experiments and Monodepth2 [5].

As can be seen in Table 4.10 casting depth estimation as a classification task lowers the accuracy with the pipeline we used. One of our hypothesis was that the simpler network was limiting the potential of the new training approach. To test this hypothesis, we experimented with a more complex network but once again achieved the same results, as shown in the last two rows of 4.10. We therefore concluded that the differences in training data augmentation and lack of left-right consistency were the cause for the different results compared to Bello et al. [11].

KITTI-Odometry	Training	LOWER IS BETTER			HIGHER IS BETTER		
		Abs Rel	Sq Rel	RMSE	$d < 1.25$	$d < 1.25^2$	$d < 1.25^3$
Our Model	MS	0.133	1.070	5.463	0.818	0.932	0.970
Our Model w/ DV	MS	0.133	1.046	5.396	0.817	0.932	0.970
Our Model	M	0.123	0.721	4.338	0.832	0.943	0.977
Our Model w/ DV	M	0.244	1.982	6.930	0.600	0.862	0.952
Our Model w/ DV w/ VO Pose	M	0.221	1.847	6.563	0.665	0.898	0.967
Complex Model	M	0.131	1.165	5.546	0.823	0.933	0.970
Complex Model w/ DV	M	0.129	0.984	5.705	0.817	0.930	0.973

Table 4.10: Results on KITTI Eigen Test Split by casting depth estimation as a classification task following Bello et al. [11] work.

## 4.5.2 Temporal Consistency

In attempt to simulate the left-right consistency term often used in stereo training we implemented a similar approach for consistency in monocular videos. The idea of the left-right consistency term is that depth maps predicted for the left and right images should be consistent, meaning that projecting the left disparity map on the right one should lead to a very small reprojection error, mostly due to occlusions. The same approach can be used when training with monocular videos by employing the pose predicted by the pose network to warp the depth maps and enforce consistency for the predictions across multiple frames. One added benefit of this approach is the generation of scale-consistent depth maps. The latter is not really useful in our pipeline since we can recover depth with the velocity loss term but it allows an easier scale recovery post-training in case that the velocity loss term was not used. In Figure 4.12, we show the pipeline modifications to allow temporal consistency. We add to our loss an additional term

$$L_{tc} = \lambda_{tc} \sum_t \sum_{i,j} |D_{t \rightarrow t'}(i, j) - D_{t'}(i, j)|$$

to enforce temporal consistency between the predicted depth maps. In Table 4.11 we show the ablation study regarding different weights for the  $L_{tc}$  term. It can

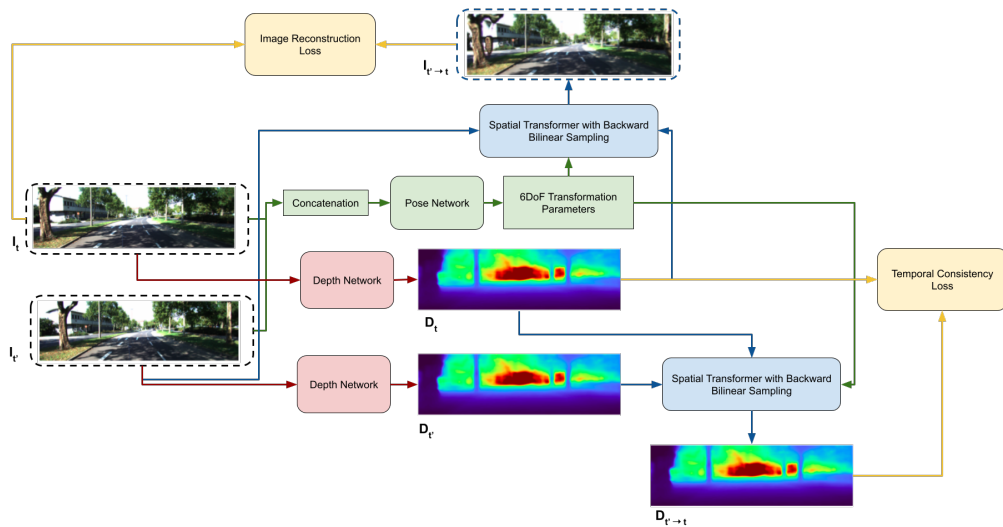


Figure 4.12: Temporal consistency pipeline with 2 input frames.

be seen that temporal consistency didn't allow us to improve accuracy while it increased the training time. For these reasons we didn't include it in our pipeline and the models trained for the results in Chapter 5 do not make use of temporal consistency.

$I_{tc}$	LOWER IS BETTER			HIGHER IS BETTER		
	Abs Rel	Sq Rel	RMSE	$d < 1.25$	$d < 1.25^2$	$d < 1.25^3$
0	0.123	0.721	4.338	0.832	0.943	0.977
0.001	0.125	0.732	4.401	0.829	0.943	0.977
0.01	0.128	0.735	4.503	0.821	0.942	0.977
0.1	0.132	0.746	4.637	0.812	0.944	0.978

Table 4.11: Results on KITTI odometry sequences 0, 4, 5, 7 for different weights for the temporal consistency loss term. Accuracy doesn't improve compared to the baseline without temporal consistency.

### 4.5.3 YUV input data

The most common image format is RGB, where each pixel color is usually encoded as 3 bytes representing the Red, Green, Blue components of the pixel color. RGB while being easy to understand and use it's not very useful when trying

to compress images by taking into account human perception. As an example, humans are more sensitive to the green color than red and blue and therefore if we downsample the green channel we will notice a much bigger difference than downsampling the red channel. The YUV image format, originally developed to maintain compatibility with black and white TVs, takes into account human perception and it allows easier compression of image data. The YUV model defines a color space in terms of one luminance component (Y) and two chrominance components, called U (blue projection) and V (red projection) respectively. The most common version of YUV image format is YUV420 where the horizontal and vertical resolution of the UV channels are halved to allow for lower data bandwidth. For these reasons, YUV image format is nowadays very common on mobile phones and the default format on Android. Since the output images of the camera on Android are in YUV format, a conversion is needed. An alternative lies in training the neural network with YUV input image data, removing the need for conversion from RGB to YUV at inference time on the mobile phone. To make sure that different image formats don't lead to lower accuracy we trained our networks with YUV input images. As can be seen in Table 4.12, the new format doesn't affect accuracy, and the differences measured are explainable as noise due to the random initialization.

KITTI-Odometry	LOWER IS BETTER			HIGHER IS BETTER		
	Abs Rel	Sq Rel	RMSE	$d < 1.25$	$d < 1.25^2$	$d < 1.25^3$
RGB	0.104	0.642	3.999	0.865	0.948	0.976
YUV	0.104	0.640	3.989	0.864	0.948	0.976

Table 4.12: Results on KITTI odometry sequences 0, 4, 5, 7 with different input image formats.

# Chapter 5

## Evaluation and Results

### 5.1 KITTI Dataset Results

Here we will present the main results achieved on the KITTI dataset [9]. When training with sparse depth maps as additional input we were able to train only using the left images of the Odometry Split since we didn't have sparse depth maps available for the other frames and images. To maintain a fair comparison we therefore trained using only this subset of data unless otherwise noted. Pose evaluation was run on the 8691 frames of the test set of the Odometry Split.

KITTI-Odometry	Training Data	LOWER IS BETTER				HIGHER IS BETTER		
		Inference Time	Abs Rel	Sq Rel	RMSE	$d < 1.25$	$d < 1.25^2$	$d < 1.25^3$
Monodepth2 †	M+S	380 ms	0.119	0.733	4.274	0.846	0.945	0.976
Monodepth2 †	M+Vel	380 ms	0.126	0.758	4.330	0.840	0.941	0.974
PydNet †	M+Vel	147 ms	0.142	0.841	4.524	0.818	0.944	0.977
Ours	M+Vel	141 ms	0.123	0.721	4.338	0.832	0.942	0.977
Ours	M+Vel+SD	152 ms	0.103	0.627	3.950	0.866	0.949	0.977

Table 5.1: Evaluation on KITTI Odometry with models trained on the KITTI Odometry Split. Legend: †: Our reimplementation, M: Monocular videos, S: Stereo, Vel: Velocity information for scale recovery, SD: Sparse depths using our simplified autoencoder.

KITTI-Eigen-Split	Training Data	LOWER IS BETTER				HIGHER IS BETTER		
		Inference Time	Abs Rel	Sq Rel	RMSE	$d < 1.25$	$d < 1.25^2$	$d < 1.25^3$
Monodepth2 w/o pretraining	M+S	-	0.127	1.031	5.266	0.836	0.943	0.974
PydNet	S	-	0.153	1.363	6.030	0.789	0.918	0.963
Monodepth2 †	M+S	380 ms	0.132	1.141	5.463	0.820	0.933	0.970
Monodepth2 †	M+Vel	380 ms	0.135	1.153	5.642	0.812	0.929	0.968
PydNet †	M+Vel	147 ms	0.145	1.306	5.749	0.802	0.924	0.965
Ours	M+Vel	141 ms	0.134	1.070	5.763	0.801	0.928	0.968

Table 5.2: Evaluation on KITTI Eigen Split with models trained on the KITTI Odometry Split. Legend: †: Our reimplementation, M: Monocular videos, S: Stereo, Vel: Velocity information for scale recovery.

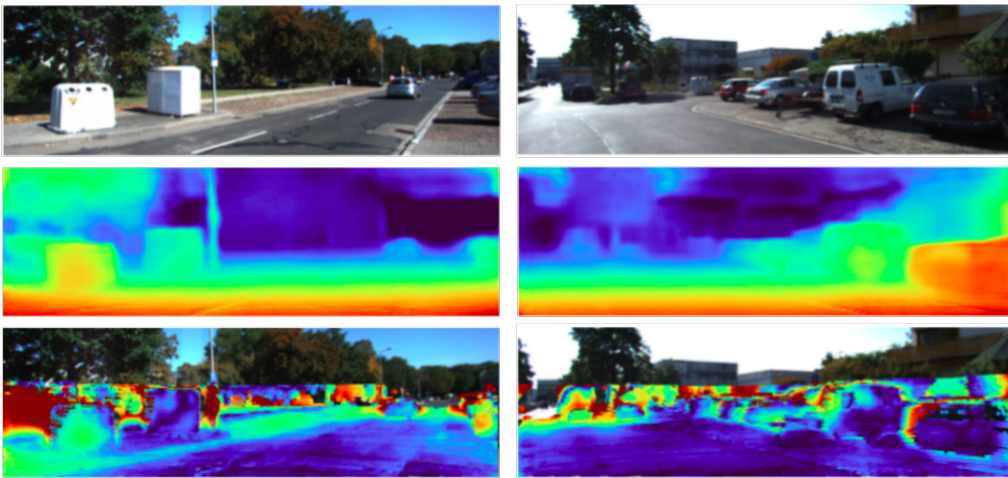


Figure 5.1: Two examples of disparity maps estimated by our network without input sparse depths. From top to bottom: input image, estimated depths maps, per-pixel absolute relative error with KITTI non-improved depth maps.

## 5.2 Stockholm Dataset

Given the different resolution compared to the KITTI dataset, the input images of the Stockholm dataset were rescaled to  $640 \times 352$  during training instead of  $640 \times 192$ . Sparse depth maps are not available for the dataset and therefore experiments with the autoencoder and sparse depths inputs will not be discussed.

Figure 5.5 shows that the network was able to learn scale since most of the points lie on the ideal red line. The qualitative evaluation offered in Figure 5.4 shows how traffic signs and poles are detected much better than when training with the

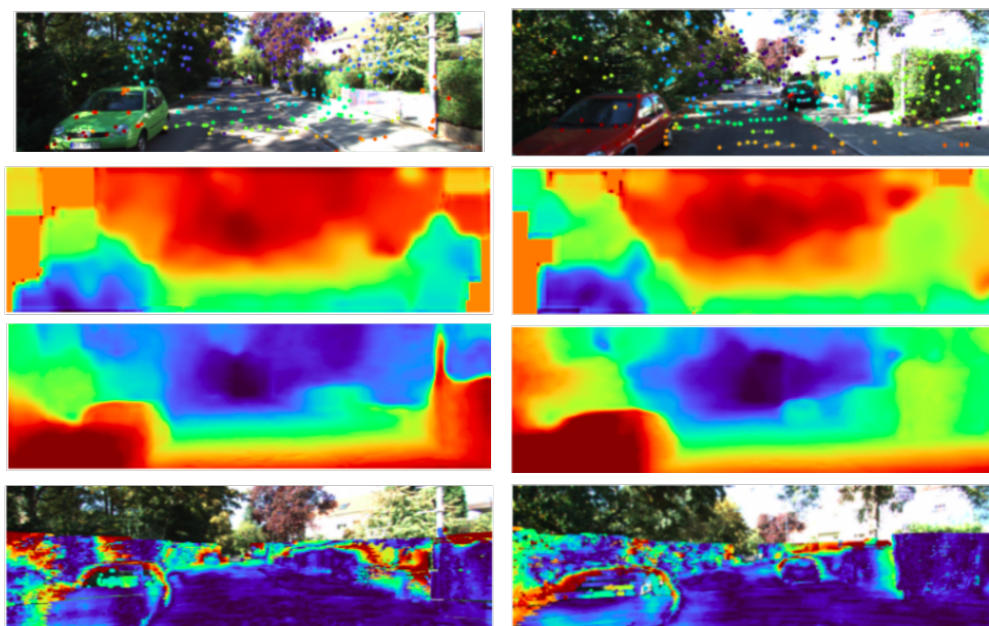


Figure 5.2: Two examples of disparity maps estimated by our network with input sparse depths. From top to bottom: input image with sparse disparity maps, output of the "autoencoder", estimated depths maps, per-pixel absolute relative error with KITTI non-improved depth maps.

KITTI dataset. It also shows the capability of detecting bridges correctly and the correct depth prediction for cars that the camera is following.

### 5.3 Pose Evaluation

The goal of our networks was not to provide accurate pose estimations. Nonetheless, learning accurate depths, with scale, using monocular videos, requires that the output pose is relatively accurate. Particularly, we expect higher accuracy of the depth maps the closer the scale of the estimated pose is to the ground truth. In Table 5.3 it is shown that more complex neural networks don't necessarily improve the accuracy of pose. This suggests the presence of a fundamental limitation

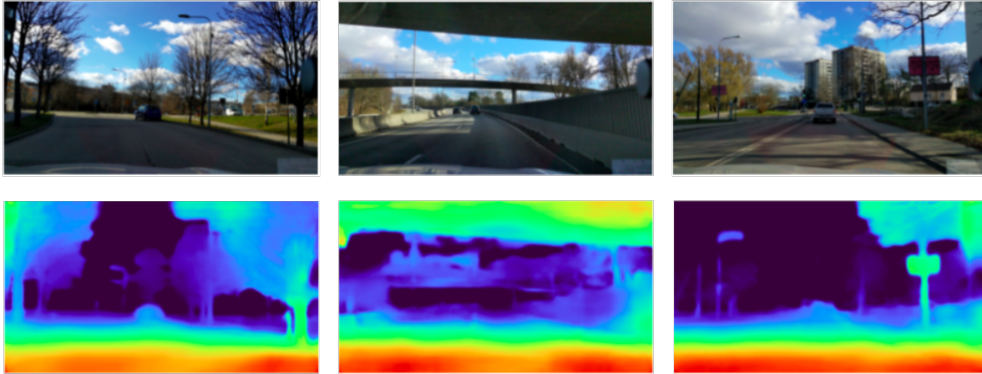


Figure 5.3: Three examples of disparity maps estimated by our network on the Stockholm dataset. It is relevant to note how the poles and traffic sign are detected much better compared to the KITTI dataset. As can be seen, cars followed by the main vehicle are still detected at a realistic distance rather than infinite depth.

in the training scheme that doesn't allow to learn more accurate poses.

KITTI-Odometry	Seq. 00		Seq. 04		Seq. 05		Seq. 07	
	$T_{rel}$	$R_{rel}$	$T_{rel}$	$R_{rel}$	$T_{rel}$	$R_{rel}$	$T_{rel}$	$R_{rel}$
MD2 PoseNet	10.059	4.081	3.536	2.347	10.926	5.212	9.958	6.964
EffNetB4	9.351	3.567	3.508	2.244	8.540	4.236	8.590	6.128
EffNetB0	12.135	4.837	3.911	2.432	12.335	5.936	11.329	8.619

Table 5.3: Results on KITTI odometry sequences 0, 4, 5, 7 with different pose networks.

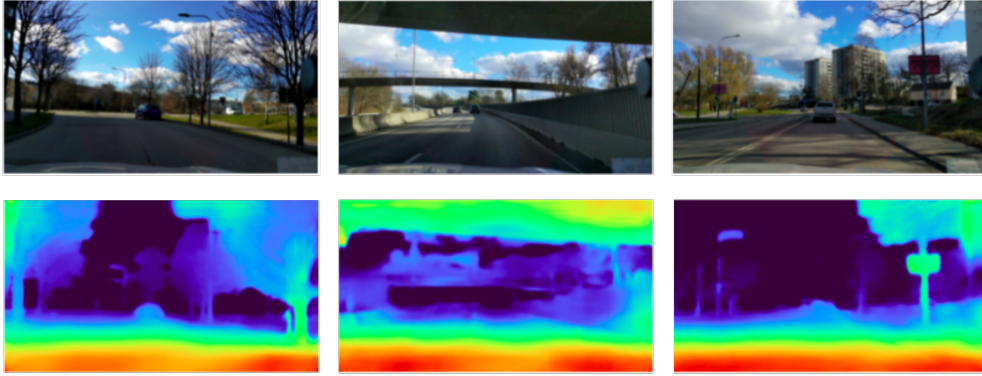


Figure 5.4: Three examples of disparity maps estimated by our network on the Stockholm dataset. It is relevant to note how the poles and traffic sign are detected much better compared to the KITTI dataset. As can be seen, cars followed by the main vehicle are still detected at a realistic distance rather than infinite depth.

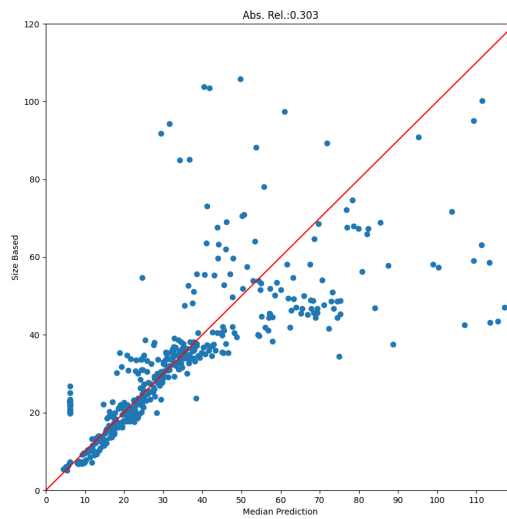


Figure 5.5: Relationship between size based object depth and median predicted depth. Aside from some outliers, particularly at higher distances, we can see that the network recovered the scale correctly since the ideal red line is fairly close to the points.

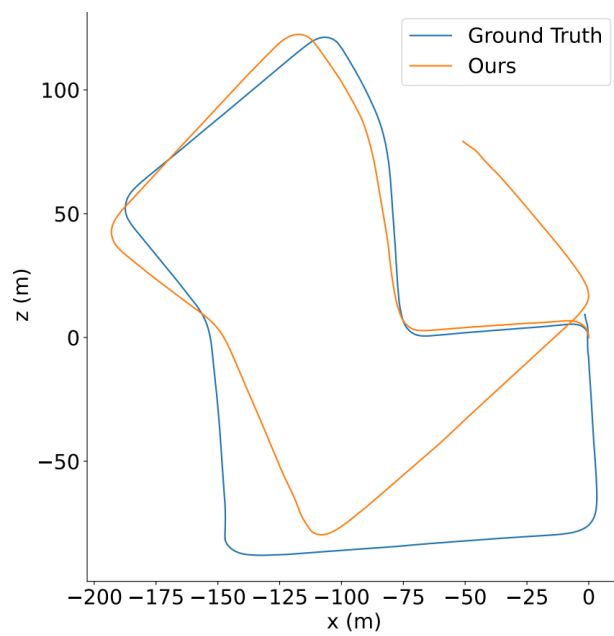


Figure 5.6: Path predicted by the Monodepth 2 (ResNet18) pose network in sequence 7 (sequence part of the Odometry Split test set).

## Chapter 6

# Conclusions and further developments

The aim of this thesis was to tackle monocular depth estimation with a lightweight neural network suitable for realtime inference on a mobile phone. Therefore, the first task was an exploration of different neural network architectures to lower inference time while maintaining accuracy. It was shown how architectures developed for image classification are not ideal for depth estimation due to the different number of channels needed at the different stages of the network. A modified version of PydNet [21], with a different number of channels, was used as encoder. The decoder of PydNet was instead simplified by employing faster nearest neighbour upsampling followed by depthwise convolutions.

The next task was to inject scale information in the monocular training, tackling the problem of prediction of depth maps only up-to-scale when not using stereo data for training. This was achieved employing ground truth velocity provided by GPS/OXTS sensors to train a correctly scaled pose and therefore depth maps.

Finally, following previous work by Andraghetti et al. [2], we improved the accuracy of the predicted depth maps using as additional input sparse depth maps estimated by a monocular SLAM algorithm. To maintain low inference time, the autoencoder from [2] was simplified without major losses in accuracy.

The system proposed allows to train easily on new data since only monocular videos and GPS information are needed. This simplifies deployment to different scenarios since it makes fine-tuning on new data much easier.

On the other hand one major limitation to deployment in the wild is the possible different camera orientation and position that can greatly affect the depth maps quality. Future developments will try to improve the robustness of the neural network to different camera focal lengths, mounting positions and orientations.

Moreover, the current training pipeline requires most of the images in the train set to be static scenes. On the KITTI dataset auto-masking allows to not predict infinite depths for cars that the camera is following. Auto-masking is not enough when most of the training data contains cars moving at speeds very similar to the camera, like on highways. This makes manual filtering of the training data required. Future work will try to remove this limit allowing the network to ignore more effectively non-static regions of the image.

# Bibliography

- [1] Filippo Aleotti, Fabio Tosi, Matteo Poggi, and Stefano Mattoccia. Generative adversarial networks for unsupervised monocular depth prediction. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11129 LNCS:337–354, 2019. 12
- [2] Lorenzo Andraghetti, Panteleimon Myriokefalitakis, Pier Luigi Dovesi, Ben Luque, Matteo Poggi, Alessandro Pieropan, and Stefano Mattoccia. Enhancing Self-Supervised Monocular Depth Estimation with Traditional Visual Odometry. *Proceedings - 2019 International Conference on 3D Vision, 3DV 2019*, pages 424–433, 2019. 12, 15, 37, 38, 39, 40, 41, 59
- [3] Irwan Bello. LambdaNetworks: Modeling Long-Range Interactions Without Attention. *ICLR*, pages 1–31, 2021. 14
- [4] Shariq Farooq Bhat, Ibraheem Alhashim, and Peter Wonka. AdaBins: Depth Estimation using Adaptive Bins. *Cvpr*, 2021. 11
- [5] Hemang Chawla, Matti Jukola, Terence Brouns, Elahe Arani, and Bahram Zonooz. Crowdsourced 3D mapping: A combined multi-view geometry and self-supervised learning approach. *IEEE International Conference on Intelligent Robots and Systems*, (1):4750–4757, 2020. 9, 13, 16, 28, 35, 48

- [6] David Eigen, Christian Puhersch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in Neural Information Processing Systems*, volume 3, pages 2366–2374, 2014. 10, 22
- [7] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep Ordinal Regression Network for Monocular Depth Estimation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2002–2011, 2018. 11, 13, 47
- [8] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research*. *The International Journal of Robotics Research*, (October):1–6, 2013. 13, 26
- [9] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012. 21, 52
- [10] Clément Godard, Oisín Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, 2017-January:6602–6611*, 2017. 12, 13
- [11] Juan L. Gonzalez and Munchurl Kim. Forget about the LiDAR: Self-supervised depth estimators with MED probability volumes. In *Advances in Neural Information Processing Systems*, volume 2020-December, pages 1–8, 2020. 13, 21, 47, 48, 49
- [12] Andrew Howard, Mark Sandler, Bo Chen, Weijun Wang, Liang Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, Yukun Zhu, Ruoming

- Pang, Quoc Le, and Hartwig Adam. Searching for mobileNetV3. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-October:1314–1324, 2019. 14, 30
- [13] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-Excitation Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(8):2011–2023, 2020. 14
- [14] Lam Huynh, Phong Nguyen, Jiri Matas, Esa Rahtu, and Janne Heikkila. Boosting Monocular Depth Estimation with Lightweight 3D Point Fusion. *arXiv*, 2020. 13
- [15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *Advances in Neural Information Processing Systems*, 2015-January:2017–2025, 2015. 12, 18
- [16] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *Proceedings - 2016 4th International Conference on 3D Vision, 3DV 2016*, pages 239–248, 2016. 11
- [17] Jun Liu, Qing Li, Rui Cao, Wenming Tang, and Guoping Qiu. MiniNet: An extremely lightweight convolutional neural network for real-time unsupervised monocular depth estimation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 166(January):255–267, 2020. 14
- [18] Fangchang Ma, Guilherme Venturelli Cavalheiro, and Sertac Karaman. Self-supervised sparse-to-dense: Self-supervised depth completion from LiDAR and monocular camera. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2019-May, pages 3288–3295, 2019. 33, 35

- [19] Raul Mur-Artal, J. M.M. Montiel, and Juan D. Tardos. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015. 15
- [20] Raul Mur-Artal and Juan D. Tardos. Visual-Inertial Monocular SLAM with Map Reuse. *IEEE Robotics and Automation Letters*, 2(2):796–803, 2017. 15
- [21] Matteo Poggi, Filippo Aleotti, Fabio Tosi, and Stefano Mattoccia. Towards Real-Time Unsupervised Monocular Depth Estimation on CPU. *IEEE International Conference on Intelligent Robots and Systems*, pages 5848–5854, 2018. 14, 31, 58
- [22] Noah S. Prywes. Man-Computer Problem Solving with Multilist. *Proceedings of the IEEE*, 54(12):1788–1801, 1966. 14
- [23] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 10425–10433, 2020. 14
- [24] Rene Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, XX(Xx):1–1, 2020. 11
- [25] Vitor Guizilini Rares, Ambrus, Sudeep Pillai, Allan Raventos, and Adrien Gaidon. 3D packing for self-supervised monocular depth estimation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2482–2491, 2020. 13, 43

- [26] Chang Shu, Kun Yu, Zhixiang Duan, and Kuiyuan Yang. Feature-Metric Loss for Self-supervised Learning of Depth and Egomotion. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12364 LNCS:572–588, 2020. 12
- [27] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. *36th International Conference on Machine Learning, ICML 2019*, 2019-June:10691–10700, 2019. 31
- [28] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pages 6565–6574, 2017. 15
- [29] Rui Tian, Yunzhou Zhang, DeLong Zhu, Shiwen Liang, Sonya Coleman, and Dermot Kerr. Accurate and Robust Scale Recovery for Monocular Visual Odometry Based on Plane Geometry. 2021. 15
- [30] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity Invariant CNNs. *Proceedings - 2017 International Conference on 3D Vision, 3DV 2017*, pages 11–20, 2018. 13, 38
- [31] Tom Van Dijk and Guido De Croon. How do neural networks see depth in single images? *Proceedings of the IEEE International Conference on Computer Vision*, 2019-October:2183–2191, 2019. 29
- [32] Yekai Wang. MobileDepth: Efficient monocular depth prediction on mobile devices. *arXiv*, 2020. 14, 33, 34

- [33] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. 20
- [34] Jamie Watson, Michael Firman, Gabriel Brostow, and Daniyar Turmukhambetov. Self-supervised monocular depth hints. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-October, pages 2162–2171, 2019. 15
- [35] Diana Wofk, Fangchang Ma, Tien Ju Yang, Sertac Karaman, and Vivienne Sze. FastDepth: Fast monocular depth estimation on embedded systems. *Proceedings - IEEE International Conference on Robotics and Automation*, 2019-May:6101–6108, 2019. 14
- [36] Nan Yang, Lukas Von Stumberg, Rui Wang, and Daniel Cremers. D3VO: Deep Depth, Deep Pose and Deep Uncertainty for Monocular Visual Odometry. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1278–1289, 2020. 15