

ETDK-dolgozat

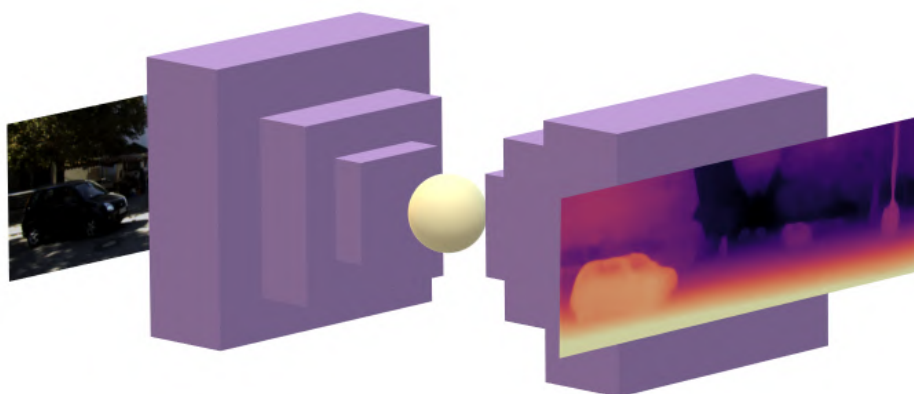
Fazakas Hunor-Attila

XXVI. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)

Informatika I.: elméleti módszerek, kísérletek, szimulációk szekció szekció

Kolozsvár, 2023. május 18–21.

Monokuláris mélységbecslés



Szerző:

Fazakas Hunor-Attila

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Témavezetők:

drd. Pável Szabolcs, tanársegéd

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

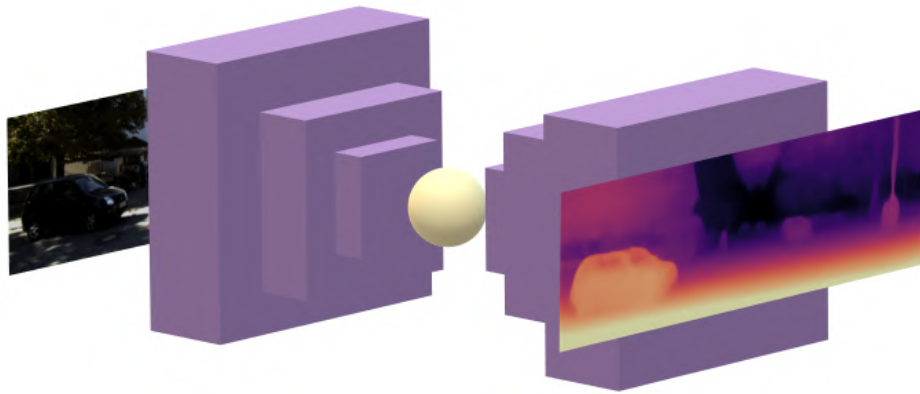
Moschidis Emmanouil, Robert Bosch SRL

**XXVI. Transylvanian Students' Scientific Conference in Formal and Empirical Sciences
(ETDK)**

Computer Science I.: Theoretical methods, experiments, simulations section

Cluj-Napoca, 18-21 May, 2023.

Monocular Depth Estimation



Author:

Fazakas Hunor-Attila

Babeş-Bolyai University, Faculty of Mathematics and Computer Science,
Computer Science, 3rd year

Supervisors:

drd. Pável Szabolcs, Assistant

Babeş-Bolyai University, Faculty of Mathematics and Computer Science

Moschidis Emmanouil, Robert Bosch SRL

Abstract

Depth estimation is a popular, and challenging computer vision task; it is important for 3D reconstruction and scene understanding. It is also a cost effective and practical alternative to LiDAR sensors in the automotive industry in applications such as assisted and fully autonomous driving.

Our main goal is to estimate a dense depth map from an input color image, where the intensity of each pixel represents depth. Since there are infinitely many acceptable depth maps that can be projected onto the same input image, we are facing an ill-posed problem. In the literature, several deep learning-based approaches have been proposed, where convolutional neural networks demonstrated potential for tackling this task.

In this paper, we review an approach that is leveraging self-supervision to perform training of a convolutional neural network without explicit ground truth. This technique is also capable of employing jointly rectified stereo image pairs and consecutive frames from videos during training. The network architecture demonstrates an encoder-decoder design that enables depth and camera pose estimation simultaneously.

Our experiments' results highlight the advantages of combining stereo and video-based sequences, as well as the benefit of self-supervised learning compared to regular supervised learning, where gathering a dataset with ground truth depth data presents a challenge. Additionally, we can observe the effect of different learning parameters, image processing strategies, and loss calculation methods on the performance of our model.

XXVI. ERDÉLYI TUDOMÁNYOS DIÁKKÖRI KONFERENCIA
CONFERINȚA ȘTIINȚIFICĂ STUDENȚEASCĂ DIN TRANSILVANIA (CȘST), EDIȚIA A XXVI-A
26TH TRANSYLVANIAN STUDENTS' SCIENTIFIC CONFERENCE (TSSC)
reál- és humántudományok • științe reale și umane • real and human sciences

KOLOZSVÁR, 2023. MÁJUS 18–21.

Kolozsvári Magyar Diákszövetség • Uniunea Studențească Maghiară din Cluj
400083 Kolozsvár, Petőfi Sándor/Avram Iancu utca 21 • tel./fax: 0755116251
e-mail: etdk@kmdsz.ro, etdk.kolozsvar@gmail.com • honlap/web: <https://etdk.kmdsz.ro>

Témavezetői igazolás

Alulírott *drd. Pével Szabolcs és Moschidis Emmanouil* igazoljuk, hogy *Fazakas Hunor-Attila* hallgató a(z) *Monokuláris mélységbecslés* című dolgozatot az alulírottak szakmai irányításával készítette el, és javasoljuk az említett tudományos munka bemutatását a XXVI. Reál- és Humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK) *Informatika I.: elméleti módszerek, kísérletek, szimulációk szekció* szekciójában.

drd. Pével Szabolcs, tanársegéd
Babeș–Bolyai Tudományegyetem, Matematika és Informatika Kar

Moschidis Emmanouil, Robert Bosch SRL

Kolozsvár, May 14, 2024

Contents

Introduction	1
1 Theoretical overview	2
1.1 Deep learning	2
1.1.1 Convolutional neural networks (CNNs)	2
1.1.2 ResNet architecture	3
1.1.3 U-Net architecture	4
1.2 Projective camera	5
1.2.1 Image disparity	5
1.2.2 Camera intrinsic parameters	6
1.2.3 Disparity to depth	7
1.2.4 Image warping	8
1.2.5 Bilinear sampling	9
2 Monocular depth estimation	10
2.1 Supervised learning	10
2.2 Self-Supervised learning	10
2.3 Self-Supervised stereo learning	11
2.4 Self-Supervised monocular learning	11
3 Implementation details	12
3.1 KITTI raw dataset	12
3.2 Data augmentation	13
3.3 Training	13
3.3.1 Warping	14
3.3.2 Loss calculation	15
3.3.3 Per-pixel minimum reprojection loss	17
3.3.4 Auto-masking	18
3.3.5 Multi-scale estimation	19
3.3.6 Explainability mask	19
3.4 Evaluation	20
3.5 Network details	21

4	Experiments and results	23
4.1	Quantitative results	23
4.2	Qualitative results	23
4.2.1	Self-supervised approach	25
4.2.2	Smoothness term modifications	25
4.2.3	SSIM loss weight modifications	26
4.2.4	Mean reprojection	27
4.2.5	Explainability mask	28
4.2.6	Auto-masking	28
5	Conclusions and further work	31

Introduction

The presence of computer vision based applications in our lives has significantly increased in the recent years due to the high interest and potential of artificial intelligence along with deep learning driven algorithms. This gives computers the capability of high-level understanding of digital images and videos, sensing and also extracting complex information into a form that can be used in other processes. Moreover, Machine Learning driven approaches are getting more and more accessible due to the rise of GPU computational power and publicly available cloud services optimised for machine learning.

The task of depth estimation is widely used in computer vision based applications such as augmented and virtual reality operations, structure from motion (Sfm) techniques and camera image post-processing. Moreover, it provides a cost effective alternative to LiDAR sensors in robotics, in terrestrial ground scanning, and in the automotive industry.

Monocular depth estimation is an ill-posed problem with an extensive literature. During recent years the research community has leveraged the advances in Deep Learning to tackle this task. There are mainly two directions in the literature regarding the adopted method, one being the supervised learning, where a dataset labeled with concrete ground truth depths is needed, and the other one being the self-supervised approach, which only demands a dataset with consecutive frames or stereo image pairs, its main objective being to learn from the similarities between a group of input frames.

In our work, we have reviewed the impact of the proposed approaches towards self-supervision (stereo versus temporal). We have also investigated several modifications in the loss function. Last but not least, we have varied several parameters and weights to gain insight with respect to their effect on the model's performance. In the next sections we present the quantitative and qualitative results of these investigations.

1 Theoretical overview

1.1 Deep learning

Deep learning is a machine learning method inspired by the humans natural learning process, it is based on teaching deep neural network models (DNNs) to make good assumptions for specific tasks, sometimes exceeding human-level performance. The term “deep” usually refers to the number of hidden layers in the neural network. These models are trained by using extensive collections of labeled or sometimes unlabeled data and various neural network architectures, which extract features automatically from the input data, eliminating the requirement for manual feature extraction.

Training a deep neural network could take an excessively long amount of time depending on the size of the dataset or even on the size and complexity of the network. However, utilizing deep learning frameworks for training neural networks can help, because they provide optimised algorithms.

1.1.1 Convolutional neural networks (CNNs)

Convolutional neural networks, commonly known as CNNs, are among the most widely used deep learning networks and are particularly effective for image processing tasks. They are able to identify patterns in the input image, such as lines, gradients, circles, surfaces and then grouping these into meaningful objects depending on their given task.

A convolutional neural network is a feed-forward neural network, having three main type of layers: convolutional layer, pooling layer and fully-connected layer.

Wood [1] reinforces that "The usage of convolutional layers in a convolutional neural network mirrors the structure of the human visual cortex, where a series of layers process an incoming image and identify progressively more complex features". **Convolutional layers** can be conceptualized as numerous small square templates known as convolutional kernels that slide across an image in search of patterns via cross-correlation. When a segment of the image aligns with the kernel's pattern, the kernel outputs a high positive value, otherwise a smaller value or zero. These layers are mainly responsible for encoding the features of an image.

Normally after an image is handled by a convolutional layer it is then passed through an **activation function** (ex. Sigmoid, ReLU), which has the effect of adding non-linearity into the convolutional neural network.

Following a convolutional layer a **pooling (downscaling) layer** is usually introduced, which has the role of reducing the previous layer's dimension, also reducing gradually the total floating point operations needed for a forward and backward pass. Like the convolutional layer, the pooling operation employs a filter that traverses the complete input, the difference being that the filter does not contain any weights. Instead, the kernel implements an aggregation function to the values within the receptive field (e.g. Max pooling, Average pooling).

The final layer of a convolutional neural network is typically a **fully-connected layer**, meaning that every output parameter is connected to all of the previous layer's parameters. This has the role of classification based on the previously extracted features from the input image. Although convolutional and pooling layers typically utilize ReLU functions, fully connected layers generally employ a softmax activation function to accurately classify inputs, generating a probability ranging from 0 to 1.

1.1.2 ResNet architecture

A convolutional neural network's complexity is determined mostly by the number and size of layers and also by the layer types. This way, the deeper a network gets, the more time and memory it will take to train the model and it will affect the inference time as well. It could also affect the overall performance of the model, because of the increase of the number of parameters introduced in the model.

Residual Network (Resnet) [2] overcomes the limitation often attributed to the effect of the so-called vanishing gradient. It is based on adopting residual blocks instead of regular convolutional blocks, the only difference being the addition of a skip connection.

When regular convolutional blocks are used, the network needs to learn an identity function, which in case of a deeper network could lead to the issue of vanishing gradient. This means that doing the back propagation, after several applications of the chain rule (due to the many layers) the gradients may decrease to zero.

$$f(x) = h(x) \tag{1}$$

The skip connections alleviate the issue of vanishing gradient by setting up an alternate shortcut for the gradient to pass through. This can be done by adding our input x to the output of the convolutional block $f(x)$ (2).

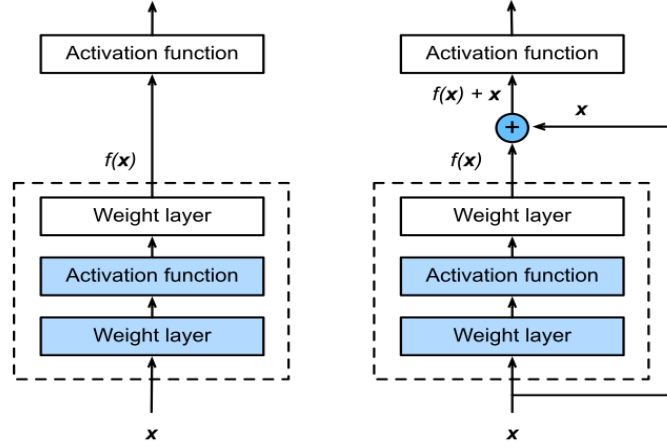


Figure 1: We can see on the left side a regular convolutional block and on the right side the proposed residual block [3].

$$f(x) + x = h(x) \quad (2)$$

Resnets can have variable sizes depending on the number of layers in a residual block or the number of blocks used in the network (Resnet18, Resnet34, ... Resnet 152). Nevertheless, there is empirical evidence that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth [2].

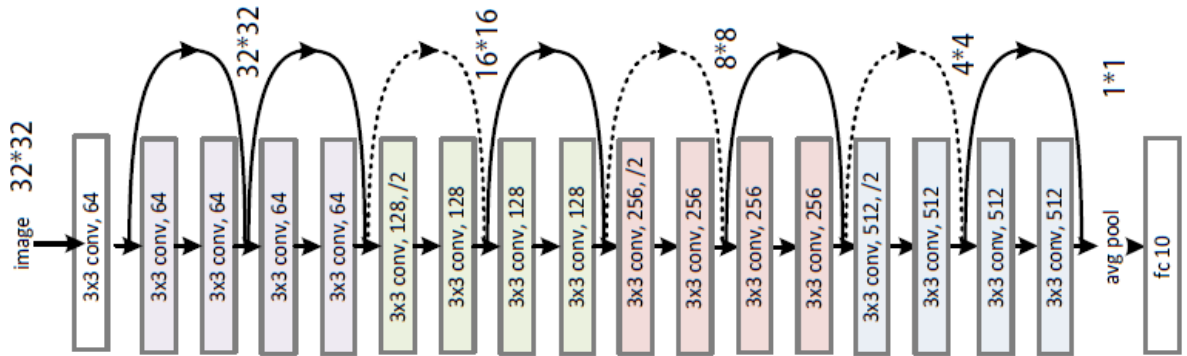


Figure 2: Resnet18 architecture. We can observe the presence of the skip connections. The dotted connections increase the number of channels and reduce the spatial dimensions [4].

1.1.3 U-Net architecture

The U-Net is a widely used architecture in Machine Learning and it was firstly introduced for the task of Biomedical Image Segmentation [5]. The architecture consists of a contracting

path (also known as encoder) which is used to capture context and features from the input image and a symmetric expanding path (decoder) that groups up features and produces a task specific output (Figure 3).

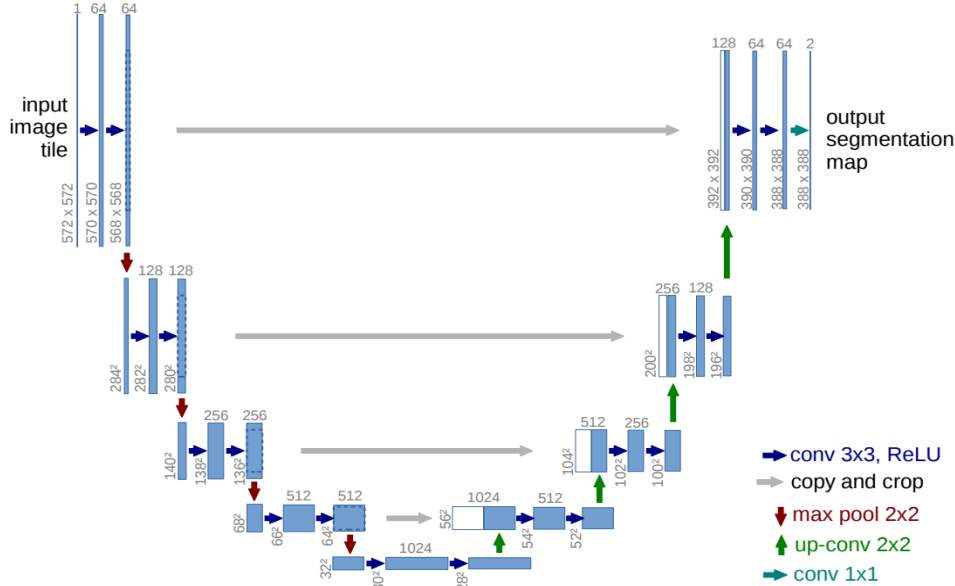


Figure 3: Example of a U-net architecture for Image Segmentation [5]. **Red** arrows indicate the down-sampling layers (Conv) and the **green** ones the up-sampling layers (UpConv). Also, with **grey** we can see the skip connections that concatenate the encoder feature map with the decoder which helps in the backward flow when calculating the image gradients.

1.2 Projective camera

1.2.1 Image disparity

We, as humans perceive the world in three-dimensions even though the input to our visual system, the images projected onto our two retinas, have only two spatial dimensions [6]. Our brain's visual system can infer depth from multiple visual signals, one such signal being the binocular disparity.

Disparity refers to the positional difference (parallax) of an object between two view perspectives. In case of human vision, the **binocular parallax** (Figure 4) results from the fact that the two eyes are laterally separated and therefore see the world from two slightly different vantage points [6]. **Motion parallax** can also contribute for calculating per-pixel disparities between an image taken in the T moment and the other one taken in the $T+1$ moment (Figure 5). This way we can take the advantage of consecutive frames from a video to extract disparities, without the need of two cameras.

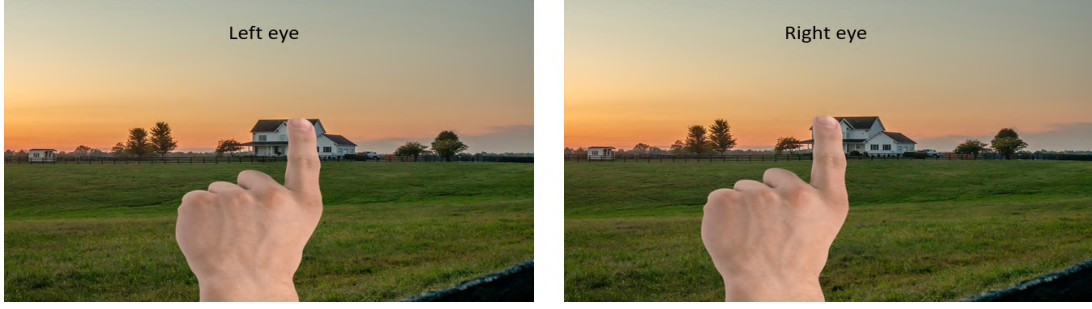


Figure 4: Binocular parallax experienced with human eyes.



Figure 5: Motion parallax from a video.

The challenges for motion-based disparity calculations rise when objects are appearing stationary in both perspectives. In these cases their disparity values will be zero. This problem originates mostly from objects moving at the same speed as the camera or could arise from entirely static scenes.

1.2.2 Camera intrinsic parameters

Every camera has a set of static parameters such as focal length, principal point, axis skew factor, field-of-view, etc. and they determine how the image was acquired. These are called intrinsic parameters and are also used in image post-processing algorithms and for image transformations as well.

The **focal length** of a pinhole camera represents the distance of the focal plane along the optical axis from the pinhole itself. Cameras with larger focal length offer increased magnification over a more restricted field of view, while smaller focal lengths offer a more extensive coverage. We can further denote f_x and f_y expressing the focal distance as a function of the pixel size in mm, meaning that for all cameras having square pixels f_x and f_y will be the same: $f_x = f_y$. The **principal point** (pp_x, pp_y) is the coordinate of the intersection of the optical axis with the focal plane.

The last parameter of the intrinsic matrix is the **axis skew factor** s , which introduces a shear transformation on the image. In most cases it has the value of 0, excepting when the image's x and y axis are not perpendicular.

$$K = \begin{bmatrix} f_x & 0 & pp_x \\ 0 & f_y & pp_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

With the help of the **intrinsic matrix** K we are able to convert points from the camera coordinate system to the pixel coordinate system (which represents our image):

$$\underbrace{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}_{\substack{\text{Pixel} \\ \text{coordinates}}} \sim \underbrace{\begin{bmatrix} f_x & 0 & pp_x & 0 \\ 0 & f_y & pp_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_K \underbrace{\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}}_{\substack{\text{Camera} \\ \text{coordinates}}} \quad (4)$$

1.2.3 Disparity to depth

Knowing the displacement of every pixel between two images from different views, we can calculate the **depth** of every pixel accordingly.

In the case of a pinhole camera, knowing the focal length f of each camera and the baseline b between them, we can calculate the depth using simple triangulation :

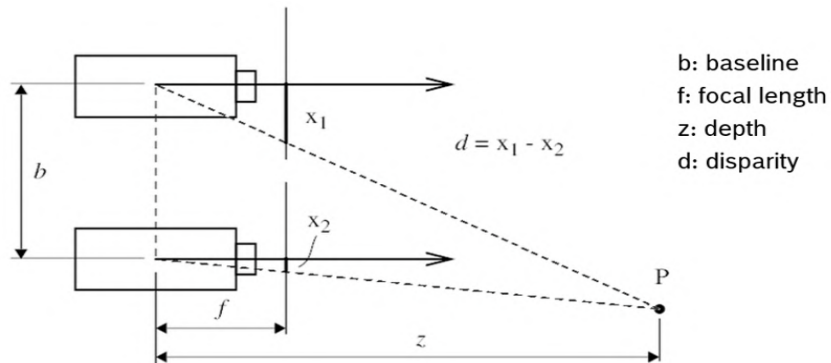


Figure 6: Triangulation method on the stereo camera system with known baseline and focal length.

Therefore, from similar triangles we can write $\frac{d}{b} = \frac{f}{z}$ from where we can directly calculate

the depth:

$$z = b * f / d \quad (5)$$

Furthermore, if we want to convert our pixel coordinates to real world coordinates, now knowing the z value of each pixel we can scale them like:

$$x = \frac{z * (u - pp_x)}{f_x}; \quad y = \frac{z * (v - pp_y)}{f_y} \quad (6)$$

1.2.4 Image warping

Warping is the operation/process of deforming a source image to match a target image, in the latter image's coordinate system. When we perform the warping, we apply calculated transformations to the images, which determine how each individual pixel will be transformed. Usually, this form of information is stored within a displacement field, which establishes the pixel positions correlation between the reference and template images [7]. Mostly, we do this when the two images have a large portion of their view in common, such as in case of two images constituting a stereo image pair.

The displacement field can be considered as a function $u : \Omega \rightarrow \mathbb{R}^d$ on the image domain Ω , where d is the dimension of the image. The generalized form:

$$T : \Omega \rightarrow \Omega; \quad T(x) = x + u(x) \quad (7)$$

Based on the definition of a displacement field (7), there are two primary approaches for performing image warping. The first one is referred to as **forward warping**, since every pixel of the old image is moved 'forward' to the new image's coordinate frame. Fundamentally, each pixel value x in the template image will be moved to a new location with $x' = T(x)$. However, the main issue with this approach stands in the transformation function being generally neither injective nor surjective. This could result in non-integer pixel values after applying the transformation, which have to be rounded. As a result, some pixels may not receive a value in the new image, while others could receive multiple values.

The second approach is based on the inverse process of the first one and it's called **backward warping**. The primary distinction is that with backward warping, a coordinate in the original image is computed for each pixel value x of the new image with the inverse of the transformation

function T^{-1} . Nonetheless, it is still possible for the function to result non-integer pixel values, but in this case we can apply an interpolation scheme (usually bilinear or trilinear) to obtain correct intensities at coordinates between pixels. Obtaining the inverse of the transformation function is frequently challenging. However, in numerous situations, an approximation can be utilized to produce appropriate outcomes, for example:

$$T^{-1}(x) \approx x - u(x) \quad (8)$$

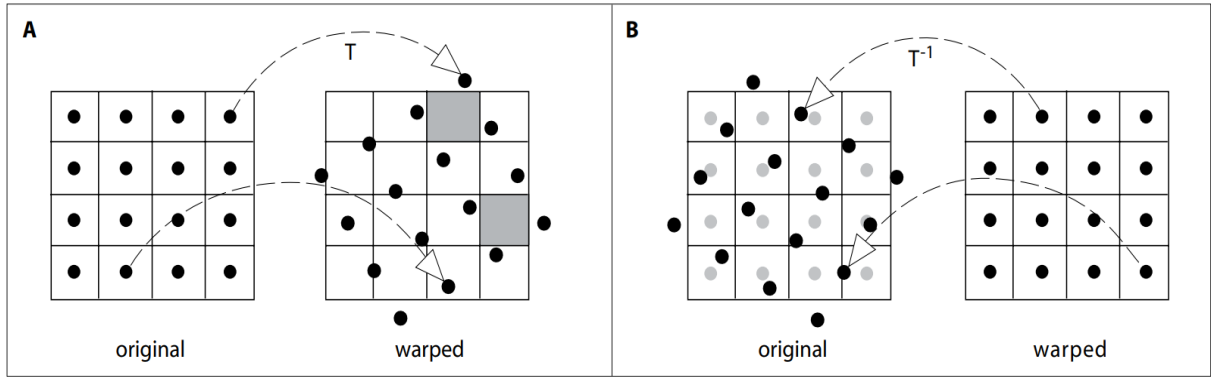


Figure 7: Forward and backward image warping. [8]

1.2.5 Bilinear sampling

Bilinear interpolation involves examining the computed location of an unknown pixel and identifying its closest 2×2 neighborhood of known pixel values. It then calculates a weighted average of these four pixels to determine the final interpolated value for the unknown pixel.

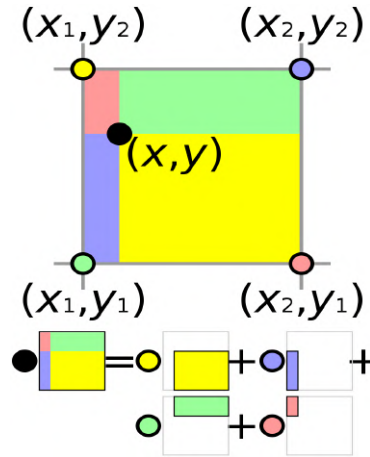


Figure 8: The 4 neighbouring pixels and the averaged new pixel's value [9].

2 Monocular depth estimation

The task of monocular depth estimation (MDE) has a wide range of approaches, most of them being deep learning-based methods. Researchers have been investigating all the possibilities, ranging from using CNNs, to Visual Transformers [10] and other different architectures. But all these approaches can be listed in two main categories regarding the learning type: **Supervised** and **Self-Supervised learning**.

2.1 Supervised learning

The first approaches presented in the literature towards tackling the task of depth estimation were supervised learning methods [11, 12, 13]. Supervised learning methods use ground truth labels after each forward pass, this way our model can understand its exact task and learn from it directly.

Translating this to our case, we would need a large dataset with rectified images from various scenarios, each one coming with its own ground truth depth map. Collecting ground truth towards generating training datasets for depth estimation typically involve the use of LiDAR sensors. However the measurements provided by most LiDAR sensors are sparse. One way to get around the effort that goes into collecting ground truth is to use a synthetic dataset for training the model [14]. However, when we talk about a large amount of data, generating a dataset with high variability which matches the real-world appearance and motion could be quite a difficult process.

2.2 Self-Supervised learning

This method is considered as an intermediate form of supervised and unsupervised learning. It does not require ground truth images for learning, it rather employs an internal representation, typically known as the supervisory signal, that is leveraged instead of ground truth during learning. In the case of depth estimation, self-supervision leverages the correspondence between a group of input images.

There are three main approaches for MDE regarding self-supervision: the first one uses rectified stereo images and takes advantage of the similarities between the left and right frame [15], the second one benefits from the consecutive frames taken from a video [16] and the third one combines these two methods [17].

2.3 Self-Supervised stereo learning

The ability to discern the shape of a scene from a single image, regardless of its appearance, is a crucial challenge in the field of machine perception. Stereo images can give an adequate supervisory signal for sensing depth, assuming the image pairs are rectified and temporally aligned. Knowing the baseline between the two cameras, we can train a model to generate pixel level correspondence between the pair of images. Although, this method looks appealing, it comes with its own limitations, the main one being that we can still observe certain artifacts at occlusion boundaries due to the pixels in the occlusion region not being visible in both images.

Firstly proposed by Xie et al. [18], stereo supervision already showed it's potential by generating the corresponding right view to the input left image, then Garg et al. [19] introduced the reconstruction loss as the main learning cue for estimating depth. Zhou et al. [16] proposed a combination of the two previous strategies that outperformed every other competing method. Introducing consistency check between the disparities predicted for both left and right views and on multiple scales [11], also contributed to the success of their work.

2.4 Self-Supervised monocular learning

Inspired by the stereo supervision, we can learn disparity from the correspondence between temporal consecutive frames. The main difference is that in order to reconstruct the synthesized image, we also need to estimate the camera pose due to the motion between the frames. This can be achieved with the presence of a separate network that predicts the ego-motion between two images. Because the image reconstruction happens only at training time, there's no need for the separate pose network when testing the model.

Firstly introduced by Zhou et al. [16], learning only from monocular sequences showed off it's weaknesses. The effectiveness of self-supervised monocular training heavily relies on the assumptions of a stationary scene and a moving camera. However, if these assumptions are violated, such as when there is object motion in the scene or the camera is stationary, the performance can be severely impacted. Godard et al.[17] found a way around this problem by introducing auto-masking, where stationary pixels got masked out, preventing contaminating the loss.

3 Implementation details

The network is implemented in Pytorch [20] and uses TensorBoard for monitoring each step of the training phase. Each strategy and approach is parameterized in a separate configuration file in order to manage clearly the different experiments. We followed the ideas introduced by Godard et al. [17] and contributed to that with trying different parameter combinations, examining their behaviour, adapted the code to the different loss calculation strategies and investigated different parameter combinations.

3.1 KITTI raw dataset

We mainly conducted our experiments on the widely used KITTI [21] dataset, which offers sequential stereo images with sparse point clouds taken by LiDAR sensors (Figure 9). Training according to the data split of Eigen and Fergus [22], we have a training split of 39,810 frames and 4,424 frames used for validation (10% of the dataset) with a resolution of 1242x375.

To ensure consistency across all images, we use the same intrinsics, setting the principal point of the camera pp_x, pp_y to the image center and the focal length to the average of all the focal lengths from the dataset $f_x = 720$. Our intrinsic matrix K is normalized before training with scaling the first row by $1/width_{image}$ and the second row with $1/height_{image}$.

During the evaluation, models which were trained only by monocular sequences will get a per-image median ground truth scaling introduced by Zhou et al. [16] before calculating the results. We skip the median scaling step for results that involve stereo supervision, as the training process already uses a known camera baseline, which allows the model to infer the appropriate scale.

Before the evaluation, we pre-processed the ground truth in order to have faster evaluation time. Projecting the ground truth point-cloud of every image for every evaluation attempt can be time consuming, hence we opted for an off-line pre-processing of the point-cloud. As established in [17], we project the 3D point-cloud of every item from the evaluation split to the image plane with the known intrinsic of the dataset and we save these in a separate file. This way, at evaluation time we can straightly compute the result between the scaled, inferred image and the loaded, pre-processed ground truth depth.



Figure 9: Raw image from the KITTI RAW dataset with the LiDAR ground truth depth converted to an image. These converted images are used for evaluating our models.

3.2 Data augmentation

Before introducing them into the network, we first apply some augmentations to the input images in order to diversify overall the loaded images. Firstly, all input images have 50% chance to get horizontally flipped or get a slight color modification. Our color modification consist of gamma, brightness and color shifts by sampling from uniform distributions in $[0.5, 2.0]$ for brightness and $[0.8, 1.2]$ for gamma and each color channel.

3.3 Training

Before starting the training, we have the ability to choose the desired configuration we want the model to be trained on. The self-supervised learning method represents the key parameter to our training and we can activate the temporal or stereo method separately or both at the same

time.

This way, the network’s input consists of images $I_t, I_{t+1}, I_{t-1}, I_s$ if we combine the two approaches, which will get encoded into the network. I_t is the target image, I_s is the stereo pair of I_t and I_{t-1}, I_{t+1} are the temporally adjacent images to I_t . We use a fully convolutional U-net for depth prediction and a pose-decoder attached to the shared encoder in order to predict the camera poses. For extracting and encoding the features from the input images, we use a shared **Resnet 18** encoder. It is possible to use a separate encoder for the pose estimation, but according to Godard et al. [17] it significantly increases training time and doesn’t bring notable increase in the results.

After the features have been encoded, we input the features of the I_t to the depth decoder, which predicts the output disparity for the target image. If we opted for temporal training, then we forward the features of I_{t+1}, I_{t-1} to the pose decoder, which then predicts the relative camera pose T between the target and the introduced images.

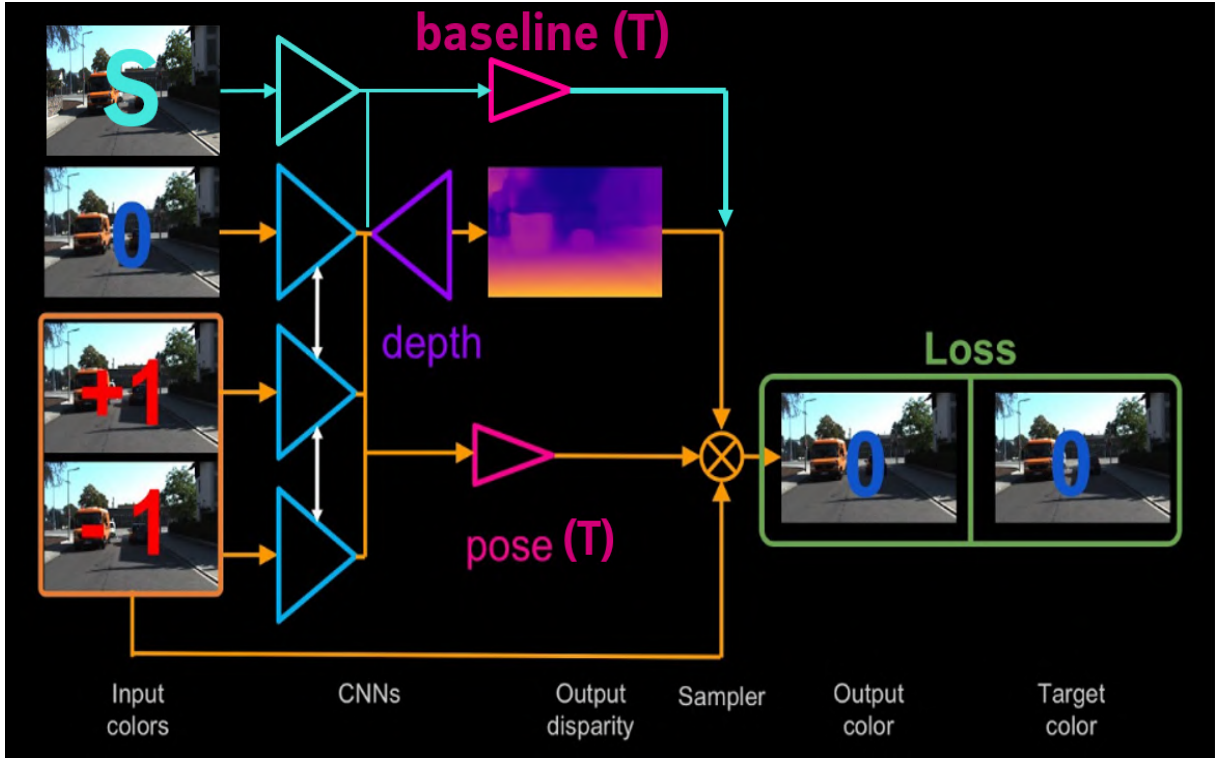


Figure 10: Combined network architecture.

3.3.1 Warping

In order to calculate the photometric losses, firstly we warp the images I_{t-1}, I_{t+1}, I_s to the target view I_t . The re-projection from $t' \in \{I_{t-1}, I_{t+1}, I_s\}$ towards I_t can be formulated as:

$$I_{t' \rightarrow t} = I_{t'} \langle \text{warp}(D_t, T_{t' \rightarrow t}, K) \rangle \quad (9)$$

where $I_{t' \rightarrow t}$ is the warped image, I_t is the source image, $\langle \rangle$ the sampling operator, $\text{warp}()$ the resulting 2D coordinates of the projected depths D_t in $I_{t'}$, D_t the depth, $T_{t' \rightarrow t}$ the relative pose of $I_{t'}$ with respect to I_t and K the pre-computed intrinsics of all the views (assumed to be identical).

Computing $T_{t' \rightarrow t}$ involves performing a translation and a rotation with an axis-angle representation. The translation matrix T and rotation matrix R (Rodrigues' rotation formula) can be written as:

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (10)$$

$$R = \begin{bmatrix} \cos\theta + u_x^2(1 - \cos\theta) & u_x u_y(1 - \cos\theta) - u_z(\sin\theta) & u_x u_z(1 - \cos\theta) + u_y(\sin\theta) & 0 \\ u_y u_x(1 - \cos\theta) + u_z(\sin\theta) & \cos\theta + u_y^2(1 - \cos\theta) & u_y u_z(1 - \cos\theta) - u_x(\sin\theta) & 0 \\ u_z u_x(1 - \cos\theta) - u_y(\sin\theta) & u_z u_y(1 - \cos\theta) + u_x(\sin\theta) & \cos\theta + u_z^2(1 - \cos\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (11)$$

where θ represents the angle around the axis u_x, u_y, u_z . These two can be summarized in a 4x4 transformation matrix by a simple multiplication: $T \times R$.

In the case of the stereo supervision we do not need any rotation for warping, because the two views are horizontally aligned with the distance of the baseline from each other. After warping, we use bilinear sampling, as described in [23], to sample the source images, this technique being locally sub-differentiable.

3.3.2 Loss calculation

If we warped all the source images to the target image, we can formulate our problem as the minimization of a photometric reprojection error at training time. Our error will be the sum of all the reconstructed source image photometric errors $pe()$:

$$L_p = \sum_{t'} pe(I_t, I_{t' \rightarrow t}) \quad (12)$$

Following [15, 24, 17] for photometric error $pe()$ we combine the L1 and SSIM [25] loss:

$$pe(I_a, I_b) = \frac{\alpha}{2}(1 - SSIM(I_a, I_b)) + (1 - \alpha)||I_a - I_b||_1 \quad (13)$$

where usually $\alpha = 0.85$. The SSIM (Structural similarity index) consists of 3 similarity functions: luminance(l), contrast(c) and structure (s), which can be written as:

$$SSIM(x, y) = [l(x, y)]^\alpha * [c(x, y)]^\beta * [s(x, y)]^\gamma \quad (14)$$

where α, β and γ are the weights of the luminance, contrast and structure terms.

Let $x = \{x_i | i = 1, 2, \dots, N\}$ and $y = \{y_i | i = 1, 2, \dots, N\}$ be the same region of the two comparable images and μ_x, σ_x^2 and σ_{xy} be in order the mean of x , the variance of x and the covariance of x and y :

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}, c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, s(x, y) = \frac{2\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \quad (15)$$

where $C_1 = (K_1L)^2$, $C_2 = (K_2L)^2$ and $C_3 = C_2/2$ are the constants that induce the numerical stability to the calculation. L represents the dynamic range which is 255 for an 8 bit colored image, and $K_1 = 0.01$ and $K_2 = 0.03$ are parameters set by the authors in an empirical way. If we consider weighting the 3 similarities equally ($\alpha = 1, \beta = 1, \gamma = 1$) then our loss can be written as:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_3)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad (16)$$

For encouraging disparities to be locally smooth, as introduced by Godard et al. [15] we apply an edge-aware smoothness term to our disparity gradients:

$$L_s = |\partial_x d_t^*|e^{-|\partial_x I_t|} + |\partial_y d_t^*|e^{-|\partial_y I_t|} \quad (17)$$

where d_t^* is the mean-normalized inverse depth which is equal to $d_t^* = d_t/\bar{d}_t$.

This way, our **final training loss** will be a combination of the masked photometric loss and the per-pixel smoothness loss:

$$L = \mu L_p + \lambda L_s \quad (18)$$

and this will be averaged over each pixel, scale and batch.

3.3.3 Per-pixel minimum reprojection loss

Godard et al. [17] introduced this term because of a high photometric error penalty to out-of-view pixels due to egomotion at image boundaries, and occluded pixels. Self-supervised depth estimation methods at that time took an average of the reprojection error from multiple source images. However, this approach lead to complications when dealing with pixels that were present in the target image but were not visible in the source images.

One way to prevent these out of view pixels to contaminate the loss is to introduce a mask for these pixels [26, 27], but this does not handle disocclusion, where average reprojection can result in blurred depth discontinuities.

Another way to circumvent this problem is to compute the minimum of all the reprojection errors as introduced in [17] instead of the average or sum (Figure 11):

$$L_p = \min_{t'} pe(I_t, I_{t' \rightarrow t}) \quad (19)$$

where L_p is our final photometric error, I_t our input image and $I_{t' \rightarrow t}$ the reprojection image.

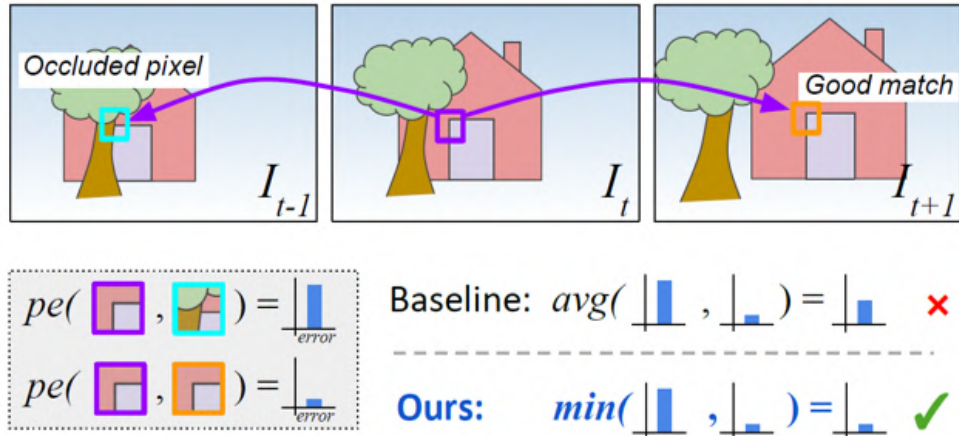


Figure 11: If the correspondences between frames are accurate, the reprojection loss will be minimal. However, when there are occlusions or disocclusions, certain pixels from the current frame may not appear in both the preceding and subsequent frames. The standard average loss technique compels the network to match occluded pixels, while the approach proposed in [17] of minimizing the reprojection loss only matches pixels with the view in which they are visible, resulting in more sharper and precised results.

3.3.4 Auto-masking

As mentioned earlier, training on monocular videos might be convenient because of the amount of data available, but there are certain noticeable disadvantages. One of the main issues is that the main assumption of a moving camera and a static scene may often be violated (e.g. when the ego-vehicle is halted) or all the objects that appear static in multiple frames (e.g. the sky or objects moving with near equal velocity to the camera). When we take into consideration all these cases where there's no motion (e.g. when the ego-vehicle is halted) or all the objects that appear static in multiple frames (e.g. the sky or objects moving with near equal velocity to the camera), we realize that our loss will have a heavy impact if we can't handle the problem. The issue can manifest as 'holes' of infinite depth in predictions while training.

In other works [16, 27, 28] they included a per-pixel mask μ to the loss. However, we opted with the mask used by Godard et al. [17], which is similar to the prior ones, but this mask has only binary values $\mu \in \{0, 1\}$. This mask is computed automatically for each forward pass by only including in the loss the pixels where the reprojection error of the warped image $I_{t' \rightarrow t}$ is lower than the error of the unwarped original source image I'_t :

$$\mu = [\min_{t'} pe(I_t, I_{t' \rightarrow t}) < \min_{t'} pe(I_t, I'_t)] \quad (20)$$

This way, with μ we can skip the stationary unwanted pixels that could contaminate the loss, and focus only on the pixels that would truly be worth training on (Figure 12).

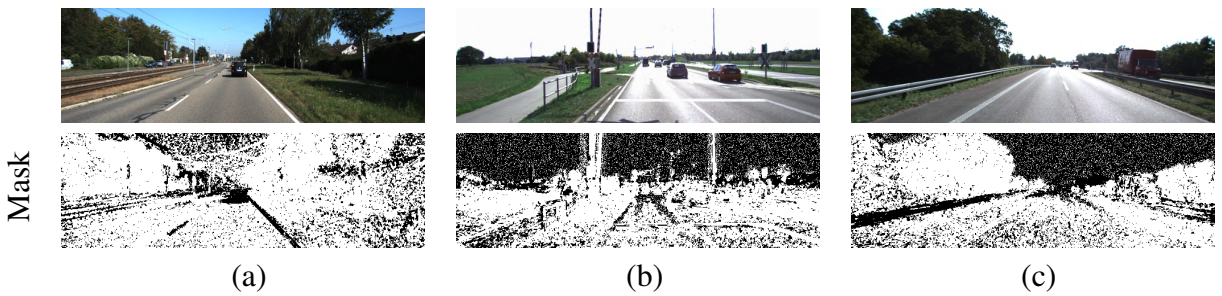


Figure 12: Visual representation of the computed masks, where white pixels are included in the final loss calculation, whereas black pixels get masked out. In the image (a) we can observe that the roadside marking is perceived as stationary, also in the (b) and (c) case the whole sky and the road barrier gets masked out.

3.3.5 Multi-scale estimation

Previous depth perception models already opted for calculating the photometric error on multiple resolution scales [19, 15] for avoiding the training objective from getting trapped in local minima. This strategy brought improvements, but generated texture-copy artifacts and holes in the inferred disparities.

As a possible solution, Godard et al. [17] proposed an improved multi-scale estimation strategy. It holds to the multi-scale term, but instead of calculating the photometric error on the intermediate scales, it decouples these by firstly upsampling the in-between disparities to the source resolution and then warps and calculates the reprojection error (Figure 13). Essentially, this approach limits the variation in depth maps across different scales, ensuring that they all contribute to the common goal of accurately reconstructing the high-resolution input image.

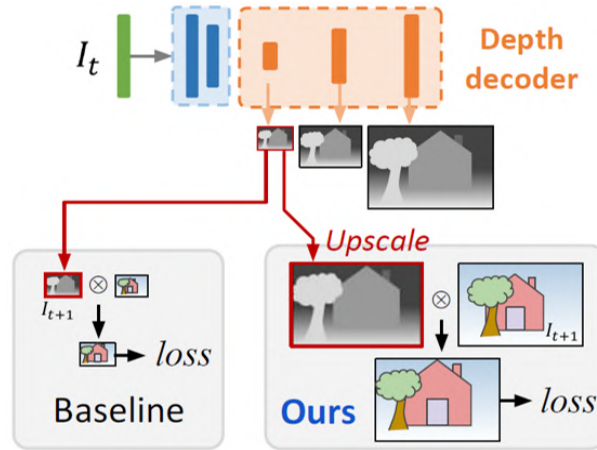


Figure 13: The baseline strategy, where losses were computed on multiple lower scales and the Upsample method [17], where in order to minimize texture-copy artifacts, they upsampled the resolution of depth predictions in intermediate scales and computed all losses based on the input resolution.

3.3.6 Explainability mask

As introduced by Zhou et al. [16], a predicted explainability mask could help in masking out regions undergoing motion, occlusions and areas where objects are not clearly visible (Figure 14).

Godard et al. [17] re-implemented this by connecting a separate decoder to the shared feature encoder and before calculating the losses it masks out the unexplainable regions predicted by the separate branch. Zhou et al. [16] also mentioned that the whole network benefits from masking

the dynamic objects with heavy motion, however the mask tends to mask out thinner structures like poles.

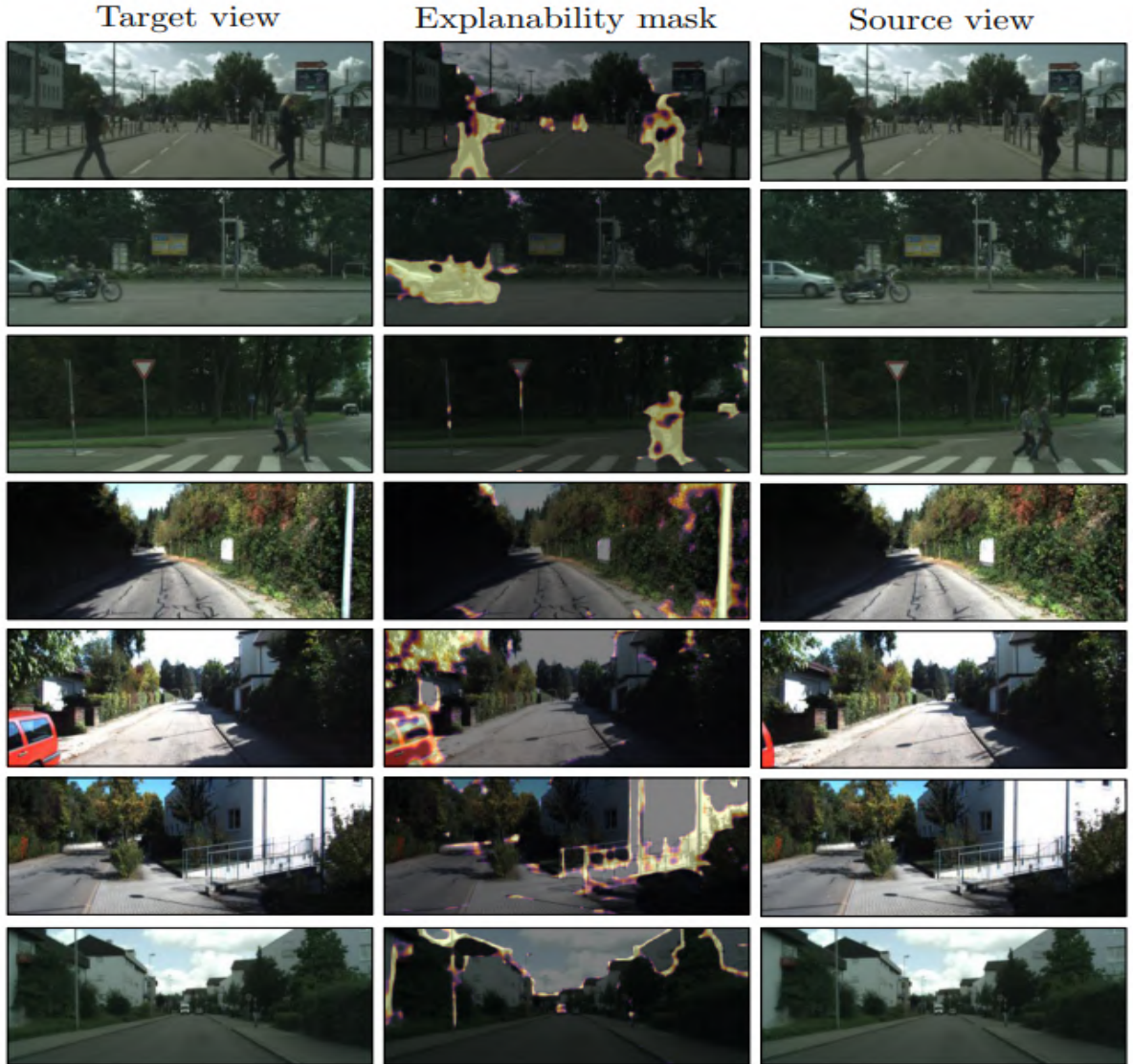


Figure 14: Visualization of some explainability masks. Unexplainable pixels are highlighted. The network predicted these mask due to motion (rows 1-3), occlusions (rows 4-5) and other factors (rows 7-8) [16].

3.4 Evaluation

We have the possibility to evaluate our saved models after the training phase or we can evaluate while training after every epoch by activating a parameter in the configuration file. The benefit of the second one is that we do not need to load in our saved model, because it is already on the GPU and we can monitorize quantitatively the whole training phase from the first until the last epoch. For our experiments we opted for the second method and used the following

evaluation metrics:

1.

$$Abs\ Rel = \frac{1}{n} \sum_{i=1}^n \frac{|D_i^* - D_i|}{D^*}$$

2.

$$Sq\ Rel = \frac{1}{n} \sum_{i=1}^n \frac{(D_i^* - D_i)^2}{D^*}$$

3.

$$RMSE = \left(\frac{1}{n} \sum_{i=1}^n (D_i^* - D_i)^2 \right)^{1/2}$$

4.

$$RMSE\ log = \left(\frac{1}{n} \sum_{i=1}^n (\log D_i^* - \log D_i)^2 \right)^{1/2}$$

5.

$$\delta_t = \frac{1}{n} |\{D_i | i \leq n, \max(\frac{D_i}{D_i^*}, \frac{D_i^*}{D_i}) < 1.25^t\}|$$

where D^* represents the ground truth and D the predicted depth map.

3.5 Network details

We use a shared encoder architecture as suggested by Godard et al. [17] which has improved training performance. For the encoder we use a Resnet18 [2] with 11M parameters. We also start the trainings with **pretrained** weights on ImageNet [29] for the encoder for improved accuracy.

All our decoders are connected to the shared encoder, meaning that overall our network is compact with relatively few parameters. Our **depth decoder** is the same as [17] using sigmoids at the output layers and ELU nonlinearities elsewhere. After converting the outputted sigmoid to depth, we constrain the values between $D_{min} = 0.1$ and $D_{max} = 100$. For the **pose-decoder** we use a smaller network with only 4 layers and ReLU activations, which outputs the axis-angle and the translation needed for the image warping. We scale these by 0.01 following Godard et al. [17]. We input into our pose-decoder the features of two temporally adjacent images from the encoder and then will get a 6-DoF relative pose output. As mentioned above, we use 3 temporally consecutive images for monocular training $I'_t \in \{I_{t-1}, I_t, I_{t+1}\}$.

Depth Decoder						
layer	k	s	chns	res	input	activation
upconv5	3	1	256	32	econv5	ELU
iconv5	3	1	256	16	↑upconv5,econv4	ELU
upconv4	3	1	128	16	iconv5	ELU
iconv4	3	1	128	8	↑upconv4,econv3	ELU
disp4	3	1	1	1	iconv4	Sigmoid
upconv3	3	1	64	8	iconv4	ELU
iconv3	3	1	64	4	↑upconv3,econv2	ELU
disp3	3	1	1	1	iconv3	Sigmoid
upconv2	3	1	32	4	iconv3	ELU
iconv2	3	1	32	2	↑upconv2,econv1	ELU
disp2	3	1	1	1	iconv2	Sigmoid
upconv1	3	1	16	2	iconv2	ELU
iconv1	3	1	16	1	↑upconv1	ELU
disp1	3	1	1	1	iconv1	Sigmoid

Pose Decoder						
layer	k	s	chns	res	input	activation
pconv0	1	1	256	32	econv5	ReLU
pconv1	3	1	256	32	pconv0	ReLU
pconv2	3	1	256	32	pconv1	ReLU
pconv3	1	1	6	32	pconv2	-

Figure 15: Detailed description of the network architecture, where **k** is the kernel size, **s** the stride, **chns** the number of channels of each layer, **res** is the downscaling factor relative to the input image’s size and **input** is the input for each layer, ↑ representing a 2x nearest-neighbor upsampling of the layer [17].

4 Experiments and results

For our experiments, we train our model following the good performing baseline established by Godard et al. [17]. We trained our model for 20 epochs with a learning rate of 10^{-4} for the first 15 epochs and then dropping down to 10^{-5} for the rest using Adam [30]. We used a batch size of 4 and our model has an input and output resolution of 640x192. For the baseline, the λ smoothness term was set to 0.001 and the α SSIM scale in the photometric error was set to 0.85. Also, we used the KITTI RAW dataset with a set of 10% of the data for validation, training on a single GPU.

4.1 Quantitative results

Parameters							Metrics						
Nr.	Method	Dfm smth	SSIM weight	Exp. mask	Mean repr.	Auto mask	Abs Rel	Sqrt Rel	Rmse	Rmse log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
1.	T+S	0.001	0.85	✓ - 0.2x	x	✓	0.121	1.333	4.880	0.204	0.881	0.952	0.975
2.	T+S	1	0.85	✓ - 0.2x	x	✓	0.126	1.343	5.001	0.208	0.872	0.950	0.974
3.	T+S	0.001	0.85	✓ - 0.2x	✓	✓	0.119	1.323	4.931	0.204	0.878	0.951	0.974
4.	T+S	0.001	0.85	✓ - 0.5x	x	✓	0.117	1.317	4.888	0.203	0.884	0.952	0.975
5.	T+S	0.001	0.55	✓ - 0.2x	x	✓	0.115	1.136	4.638	0.199	0.882	0.953	0.976
6.	T	0.001	0.85	✓ - 0.2x	x	✓	0.141	1.611	5.213	0.219	0.853	0.945	0.972
7.	T+S	0.001	0.85	✓ - 0.2x	x	x	0.116	1.463	4.863	0.200	0.889	0.954	0.976
8.	T+S	0.001	0.55	✓ - 0.5x	x	x	0.122	1.640	5.031	0.205	0.882	0.952	0.975
9.	S	0.001	0.55	x	x	✓	0.114	1.059	4.610	0.203	0.882	0.951	0.974
10.	T+S	0.001	0.85	x	x	✓	0.117	1.153	4.712	0.202	0.879	0.952	0.975
11.	T+S	0.01	0.85	✓ - 0.2x	x	✓	0.134	1.284	5.074	0.215	0.855	0.947	0.974
12.	S	0.001	0.85	✓ - 0.2x	x	✓	0.112	1.174	4.700	0.201	0.887	0.952	0.974
13.	T+S	0.001	0.55	x	x	✓	0.115	1.238	4.764	0.201	0.885	0.953	0.975

Table 1: Results after evaluating each model. **T** meaning temporally supervised, **S** stereo supervised and **T+S** combined. Also, **teal** cells represent the parameters changed compared to our baseline [17] (1st row) and the cells with **lime** are our similarity metrics.

4.2 Qualitative results

Our goal is to determine the effects and benefits of the different approaches and strategies mentioned above. By understanding and examining these we can further contribute to a possibly better performing depth estimation model.

Importantly, all the **bolded** parameter values from below represent the values for our baseline model, and the normal values denote our tweaked values. Also, each image label contains the number of the experiment from the table shown above.

For our experiments, we set up a baseline from Godard et al. [17] with the explainability mask included (Figure 16), and tweaked the different parameters in order to discover each one’s impact on the final model. Firstly, we started our line of experiments with examining the different **self-supervised approaches** (T, S and T+S), then altered the **smoothness term** (0.001 and 1). Afterwards, we modified the **weight of the SSIM** (0.85 and 0.55), then tried using the **mean of the photometric losses** instead of per-pixel minimum (**min.** and mean). Finally, for our last two strategies, we examined our **explainability masks** behaviour turning it off and on with weight of 0.2 and 0.5 (**0.2**, 0.5, off), then we analyzed the impact of our **auto-masking** strategy (**on** and off).

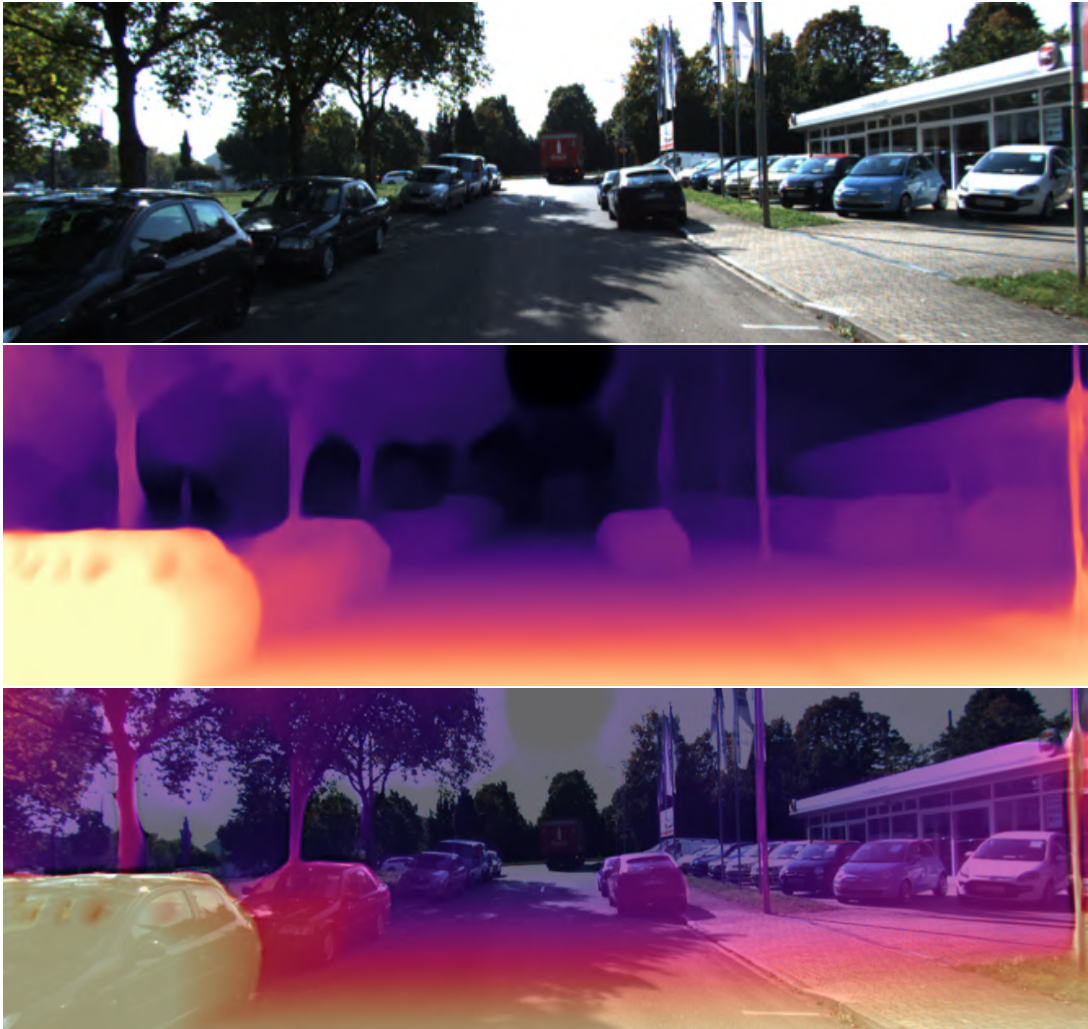


Figure 16: Dense depth map predicted by our baseline model, overlaid on the original input image.

4.2.1 Self-supervised approach

As we discussed earlier, the self-supervised approaches have the biggest role in our model. For our first set of experiments, we wanted to experience which approach has the most significant impact on our model.

We tried out all the three cases (T, S, T+S) and analyzing the results, we can affirm that the combined (T+S) model truly lives up to our expectations. It overall produces sharper, more detailed results and has generally more consistent behaviour when unexpected objects appear in the images. Whereas, in our only stereo model (S) we observe that thin objects like taller poles and road signs are ignored from the inferred depth maps. On the other hand, in the case of our temporal model (T) we denoted that a grainy effect appears usually on the side of the predicted images. Also, in this case we saw that the thinner objects (stoplights, road signs) were shaped out much better then in our other models (Figure 17).

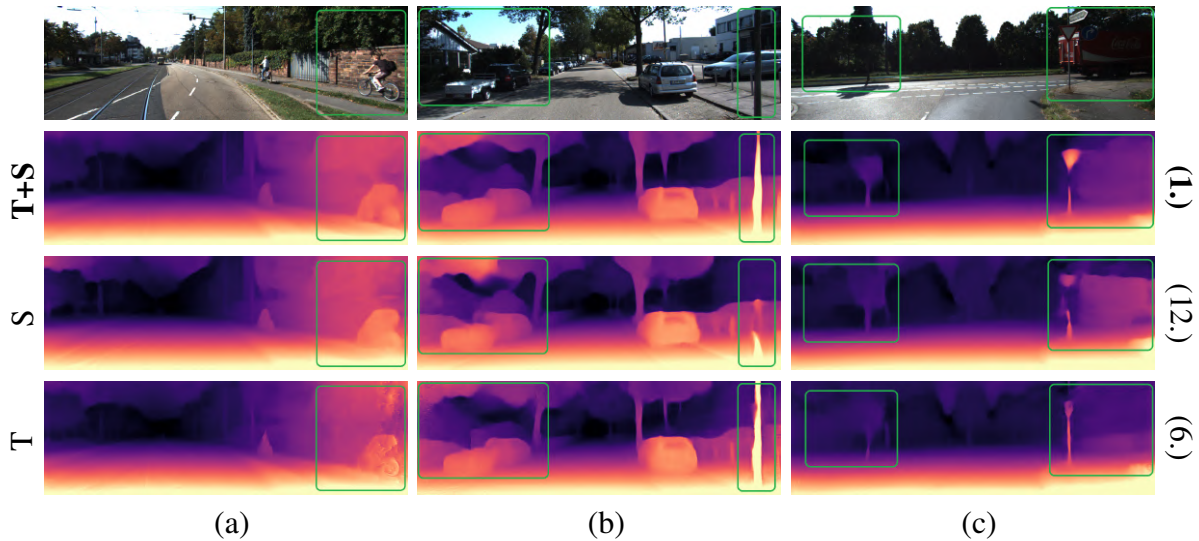


Figure 17: Visual results of the different self-supervised approaches. For the first picture (a) we can encounter the grain effect produced by our temporal model (**T**), almost blending the cyclist with the background. In the column (b) we can visualize that our stereo model (**S**) almost completely skipped the whole pole from the result. In our last image (c) our temporal model outperforms the other two when we analyze the contour of the road sign. However, the truck behind the sign is predicted better in the stereo (**S**) case.

4.2.2 Smoothness term modifications

With the goal of forcing the disparities to be locally smooth, the edge-aware term is introduced by penalizing the image gradients. This has the benefit of smoothing out the object

edges and depth discontinuities with keeping the overall shape and detail of the object.

We examined the effect of this by comparing the commonly used $\lambda = 0.001$ and a more extreme value $\lambda = 1$ (18).

Firstly, we affirmed that indeed we get continuous depths on the objects in both cases. Secondly, we observed that the model trained with bigger smoothness tends to widen the thin structures like poles and fences, also being more confident in contouring them out. However, it tends to join sometimes two separate objects with similar depth values. Also, we observed differences in the depth values on the entire surface of objects that were closer (Figure 18).

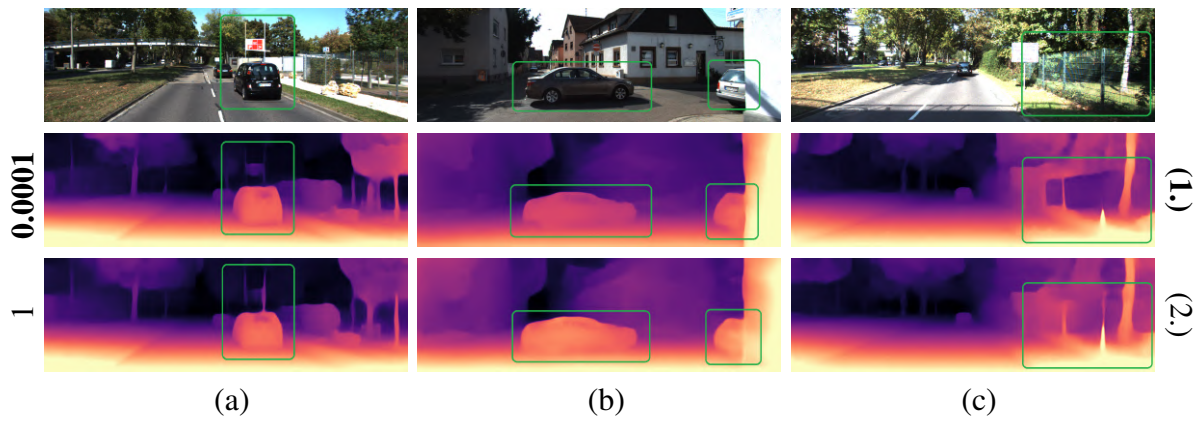


Figure 18: Visual results of the different smoothness term weights. In the column (a) we can see that in the smooth case the top of the car is inferred as connected with the sign, presumed as the same object. In the column (b) we can visualize the difference in the color intensity (depth) for both the cars. In the last column (c) we observe that our smooth model picked up the contour of the thin fence quite well, whereas our baseline model almost completely omitted it.

4.2.3 SSIM loss weight modifications

The Structural Similarity Index was already used in the first self supervised depth estimation networks [15] and is still representing a relevant part in our photometric loss (13). Since then it is still mostly used with $\alpha = 0.85$ for the weight of SSIM and $1 - \alpha$ for the weight of L1 loss.

We wanted to investigate our model's behaviour with a slightly lower weight for the SSIM loss, while still keeping it bigger than the weight of its pair, the L1 loss. We selected the following settings: $\alpha = 0.55$ for SSIM and $1 - \alpha = 0.45$ for L1 (Figure 19). For our first remark, we experienced better performance in shaping out thinner, more detailed structures for our model with modified weights. Some times it even infers the contour of the smaller road signs better than our baseline. However, analyzing carefully all of our results, we concluded that the

baseline settings produce better qualitative results with less artifacts and noise, including better predictions for bigger signs.

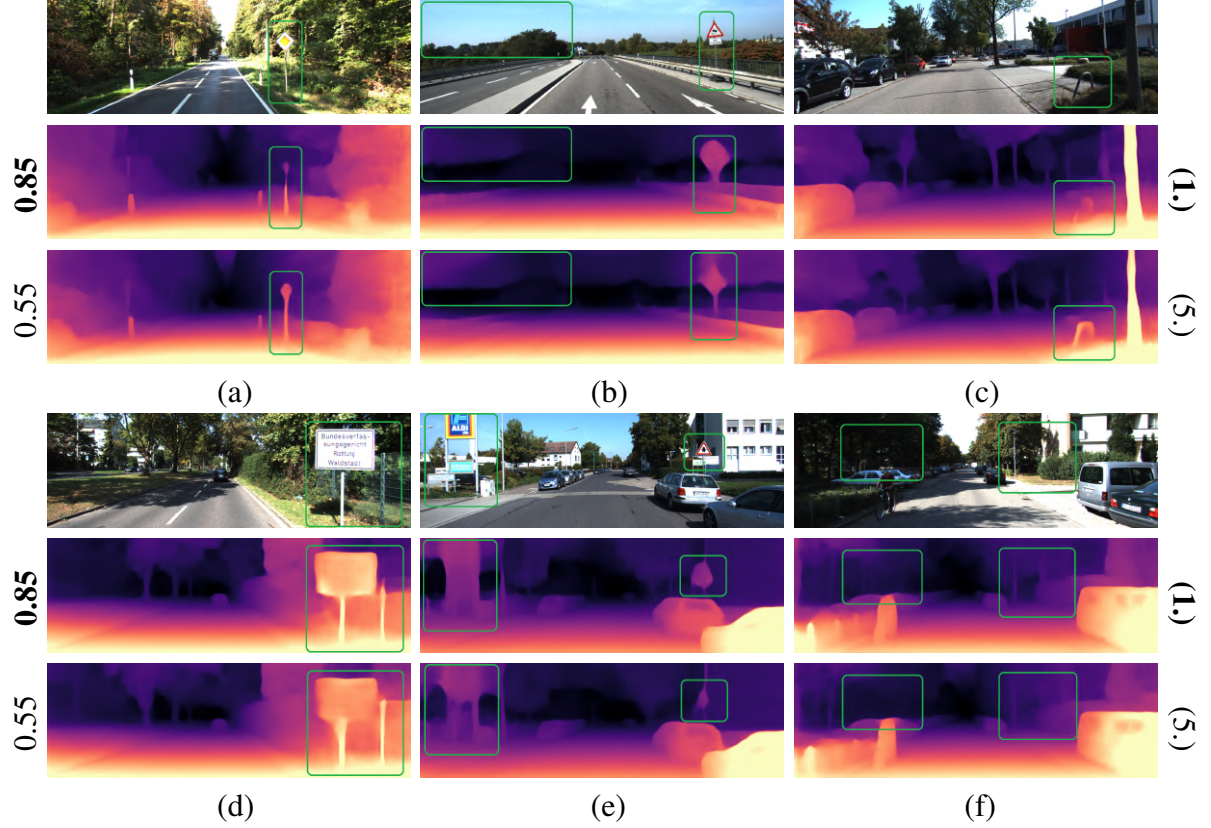


Figure 19: Visual results of the different SSIM weights. In the inferred images (a) and (c) we can see that our weight modified model shapes out precisely the thinner objects (road sign and small pipe). Also, in (d) and (e) we can observe the noise it produces around the bigger signs. In (f) our baseline model infers far more detailed representation around the trees in the background. However, in the case of (b) our baseline contoured the sign’s edges more confident and sharp, but round, whereas our modified model tries to predict it as a triangle, but inserts noise above. Furthermore, we can see on image (b) that in both cases the sky gets a depth value (shouldn’t be happening), but the modified model amplifies it, deriving to an overall worse result.

4.2.4 Mean reprojection

One other strategy for our loss calculation is taking the mean of all the reprojection losses for t' instead of the minimum (Figure 20). This way, our photometric loss calculation can be formulated as:

$$L_p = \frac{1}{N} \sum_{t'} pe(I_t, I_{t' \rightarrow t}) \quad (21)$$

where N is the number of views the photometric loss is calculated on. We can immediately observe that our model trained using the mean reprojection loss is smoother overall, resulting

in less detail over all the objects. Also, because of being smoother, it tends to blend the object boundaries with the background, causing the shapes to become indistinguishable, creating more artifacts in the inferred images and leading to a worse performance than our baseline model.

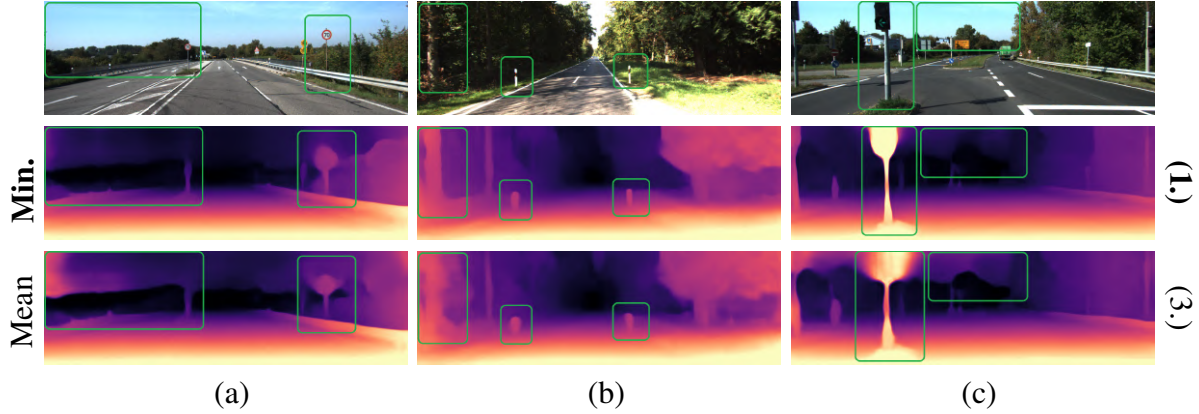


Figure 20: Visual results of the different photometric loss calculations. In the images (a) and (c) we can observe that the sign and the indicator’s shape gets washed out into the background sky. In both cases the sky also gets a depth value which induces more anomalies at further objects. In (b) the degradation in detail is visible at the sticks at the edges of the road.

4.2.5 Explainability mask

As mentioned above, the explainability mask helps in masking out objects that are unexplainable for the network and could potentially lead to artifacts in the predictions. We used it in the baseline with a weight of 0.2 for the predicted mask.

In order to investigate the real impact of the strategy, we are comparing 3 different models: the first one being our baseline, trained with a weight of 0.2 for the explainability mask, the second one having a bigger, 0.5 weight for the mask and the last one without having a separate mask predicted (Figure 21).

We observed that in all three cases the result were quite similar. However, our model without the predictive mask is more confident in predicting bigger signs and thinner structures like poles and trees. Also, dynamic objects often got fully masked out by the explainability mask.

4.2.6 Auto-masking

Stationary pixels can have a negative impact on loss calculation when talking about temporal supervision. As mentioned above, with an inexpensive method we can provide a mask in order to skip these contaminating pixels (20).

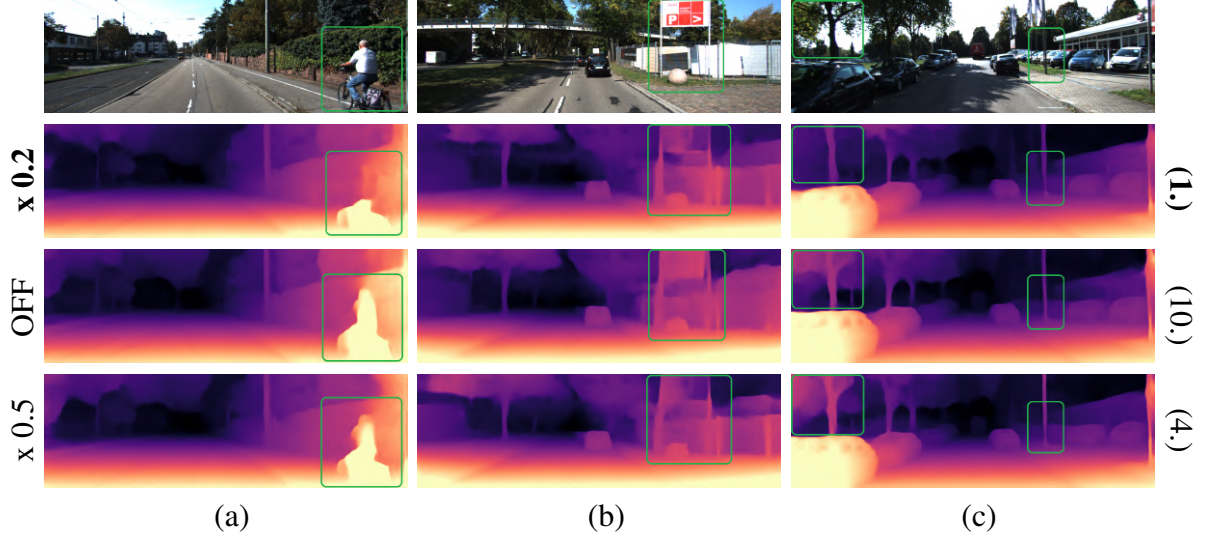


Figure 21: Visual results of the different explainability mask weights. In the (a) case we can observe that our model without the mask defines correctly the shape of the cyclist, while the other models seem to struggle, our baseline with $x0.2$ completely skipping it. In the image (b) we can remark the noise on the sign when the masks are enabled. Also, in column (c) the pole gets shaped out more clearly in the bottom when there's no mask present.

In our experiments we observed that our model with the mask turned on was performing way better on predicting thinner and smaller objects, also being more consistent overall. Also, we can observe that at sequences where the sky is occupying a good portion of the image, we can see that our model with the auto-mask is predicting notable depth values for the pixels representing the sky (incorrect, they should have maximum depth). However, when we look at our model without the mask, we can denote that the sky is inferred correctly among almost all sequences (Figure 22).

After analyzing the results, we found out that when using the auto-mask, the sky also gets masked out (due to being seen as a stationary pixel). This way the network doesn't have the chance to learn on pixels representing the sky, resulting in poor depth estimations for those regions.

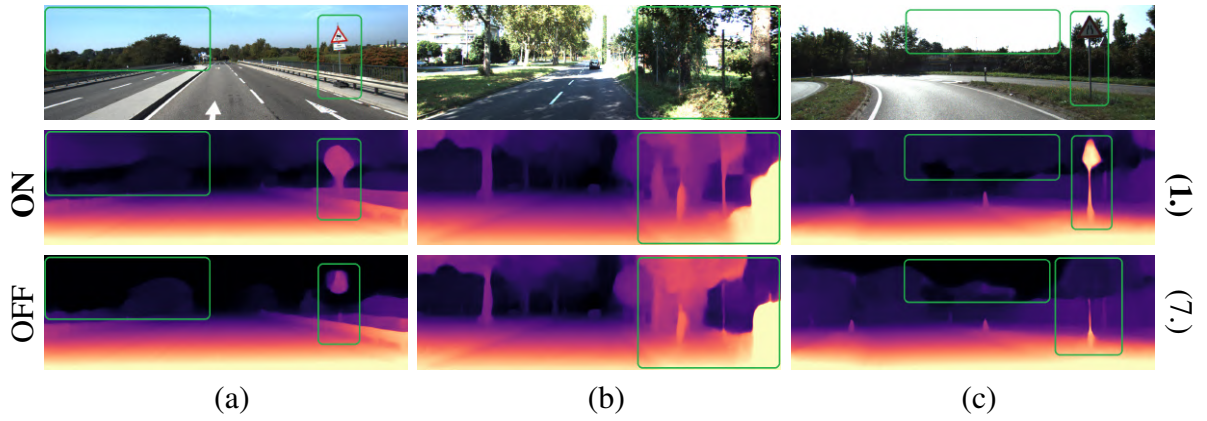


Figure 22: Visual results of the use of auto-mask. The (a) and (c) images show the poor estimation on the road sign with our model without auto-mask. Also, in the (b) case we can deduce that the fence looks very strange in both predictions, but our auto-mask model does a slightly better job. Moreover, in the (a) and (c) images we can observe the difference in the depth values predicted for the sky.

5 Conclusions and further work

In our work, we focused on examining different strategies used for training a model which can predict a dense depth map given a single RGB image. These strategies involved different learning methods, loss calculation approaches and training parameter values.

We analyzed the hypothesis of every strategy, consulting every detail regarding its implementation and theoretical proof. Then, we established a baseline configuration in order to have a starting point when we perform our experiments with different parameters.

Afterwards, we started our experiments changing one strategy or parameter value at a time compared to our baseline configuration. After we finished the experiments, we inspected and compared our results with our baseline in order to see the impact of the proposed strategies. We can affirm, that there were some cases when our quantitative results were not reflecting our qualitative comparisons, but based on our intuition, the reason could be the LiDAR depth pointcloud, being really sparse and lacking points from the top part of the image, which could impact our quantitative results when evaluating our models.

Our experiments revealed the effects of each strategy, and we can clearly affirm that the combined (T+S) self-supervised model performs better than the separate ones. Also, we can state that each loss calculation strategy has its advantages and its drawbacks as well, making it difficult to draw direct conclusions from the results.

In conclusion, as future investigations we aim to combine multiple strategies and also try out new methods with new network architectures with the goal of training a better performing depth estimation model.

References

- [1] Thomas Wood. *Convolutional Neural Network*. URL: <https://deeptai.org/machine-learning-glossary-and-terms/convolutional-neural-network>.
- [2] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pages 770–778.
- [3] Zack C. Lipton Aston Zhang, Mu Li, and Alex J. Smola. *Dive Into Deep Learning*. URL: https://d2l.ai/chapter_convolutional-modern/resnet.html.
- [4] Edouard Duchesnay. *Convolutional Neural Network*. URL: https://duchesnay.github.io/pystatsml/deep_learning/dl_cnn_cifar10_pytorch.html.
- [5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer. 2015, pages 234–241.
- [6] Ning Qian. “Binocular disparity and the perception of depth”. In: *Neuron* 18.3 (1997), pages 359–368.
- [7] Jan Modersitzki. *Numerical methods for image registration*. OUP Oxford, 2003.
- [8] Loren Arthur Schwarz. “Non-rigid registration using free-form deformations”. In: *Technische Universität München* 6.8 (2007).
- [9] *Bilinear interpolation*. URL: https://en.wikipedia.org/wiki/Bilinear_interpolation.
- [10] Md Awsafur Rahman and Shaikh Anowarul Fattah. “DwinFormer: Dual Window Transformers for End-to-End Monocular Depth Estimation”. In: *arXiv e-prints* (2023), arXiv–2303.
- [11] David Eigen, Christian Puhrsch, and Rob Fergus. “Depth map prediction from a single image using a multi-scale deep network”. In: *Advances in neural information processing systems* 27 (2014).
- [12] Huan Fu et al. “Deep ordinal regression network for monocular depth estimation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pages 2002–2011.
- [13] Iro Laina et al. “Deeper depth prediction with fully convolutional residual networks”. In: *2016 Fourth international conference on 3D vision (3DV)*. IEEE. 2016, pages 239–248.
- [14] Nikolaus Mayer et al. “What makes good synthetic training data for learning disparity and optical flow estimation?” In: *International Journal of Computer Vision* 126 (2018), pages 942–960.

- [15] Clément Godard, Oisín Mac Aodha, and Gabriel J Brostow. “Unsupervised monocular depth estimation with left-right consistency”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pages 270–279.
- [16] Tinghui Zhou et al. “Unsupervised learning of depth and ego-motion from video”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pages 1851–1858.
- [17] Clément Godard et al. “Digging into self-supervised monocular depth estimation”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pages 3828–3838.
- [18] Junyuan Xie, Ross Girshick, and Ali Farhadi. “Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks”. In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer. 2016, pages 842–857.
- [19] Ravi Garg et al. “Unsupervised cnn for single view depth estimation: Geometry to the rescue”. In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VIII 14*. Springer. 2016, pages 740–756.
- [20] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019).
- [21] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for autonomous driving? the kitti vision benchmark suite”. In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pages 3354–3361.
- [22] David Eigen and Rob Fergus. “Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pages 2650–2658.
- [23] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. “Spatial transformer networks”. In: *Advances in neural information processing systems* 28 (2015).
- [24] Hang Zhao et al. “Loss functions for image restoration with neural networks”. In: *IEEE Transactions on computational imaging* 3.1 (2016), pages 47–57.
- [25] Zhou Wang et al. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE transactions on image processing* 13.4 (2004), pages 600–612.
- [26] Reza Mahjourian, Martin Wicke, and Anelia Angelova. “Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pages 5667–5675.

- [27] Sudheendra Vijayanarasimhan et al. “Sfm-net: Learning of structure and motion from video”. In: *arXiv preprint arXiv:1704.07804* (2017).
- [28] Chenxu Luo et al. “Every pixel counts++: Joint learning of geometry and motion with 3d holistic understanding”. In: *IEEE transactions on pattern analysis and machine intelligence* 42.10 (2019), pages 2624–2641.
- [29] Olga Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: *International journal of computer vision* 115 (2015), pages 211–252.
- [30] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).