University of Amsterdam
Informatics Institute

MSc Artificial Intelligence
Master Thesis

Transformer-based guidance of Self-Supervised depth estimation

by
Bob Borsboom
10802975

September 7, 2021

Number of Credits: 48
November 2020 - September 2021

Supervisor:
Elia Bruni
Oscar Lighthart

Abstract

This thesis we present a novel approach where a transformer model is combined with a self-supervised depth estimation model. Specifically, self-attention maps from an object detection transformer model are used for two loss functions. The first loss function tries to improve the depth estimation for objects further away in the scene. The depth estimation of these regions is difficult since objects further away from the camera are sparse. With this loss, we pay extra attention to these sparse objects. The second loss function tries to improve the depth estimation for transparent/reflective objects as this is a common difficulty in depth estimation. At last the architecture of a state-of-the-art self-supervised depth estimation model is updated where a part of the convolutional layers are replaced with a transformer block. This transformer block has a global receptive field and is beneficial for dense predictions like depth estimation.
A Github repository is provided with code and instructions at: `https://github.com/boeboska/depth_estimation_thesis/tree/main/monodepth2`

Acknowledgements

The process of writing the thesis in the last months has been a fun and educational time. From the beginning, the feeling of writing my thesis at Aigritec has been good and my expectations have come true. I want to thank my supervisors for the weekly meetings we had. Every week we had open discussions about the progress, possibilities, and feedback. This really helped me continuing making progress and keeping my motivation high. At last, I want to thank Aigritec for making use of their server, this has been really helpful during the thesis period.

Contents

Chapter 1

Introduction

Humans have a rich structural understanding of the world through our past visual experience. We learned this experience through millions of observations by navigating through the world. We understand regularities of the world like roads are flat, buildings are straight and cars are supported by roads (33). Due to these observations, humans are able to estimate depth quite well. Having a good depth model can support several computer vision tasks. For example image reconstructing (25), object recognition (24), semantic segmentation (11), human pose estimation (26) and autonomous driving (13). The first models which were able to estimate depth from color images were based on lidar sensors (16). The disadvantage is that collecting large and varied training datasets with good depth information from lidar for supervised depth estimation is a challenge. To overcome this limitation, recently several self-supervised depth estimation models showed that training a depth model without lidar information (13) (6), (33) is possible. First, these models use stereo pairs (13) where the distance between images is known (explanation will follow). But since stereo pair datasets are rare this is not ideal. More recently self-supervised depth estimation with monocular videos came available (13), (6), (33). With this addition, training a depth estimation model with a self-made video is possible with good results (19). (19) Showed that it is possible to train on a self-made dataset where they used a phone camera while riding on a bike and still perform well. An important note is that this dataset is visually closely related to the KITTI data set. (17) Managed to perform depth estimation on videos taken from YouTube where the videos are made with a handheld camera while walking.
There are two subjects where the current self-supervised depth estimation models are having difficulties. First, the current self-supervised depth estimation models are having difficulties in accurately predicting depth for small objects. Most of the small objects are not captured in the depth image, see figure 1.1 for an example. There are two reasons why this is a challenge for these models. First, most small objects are further away from the camera. Since they contain not a lot of pixels the model pays less attention to these objects. As second, the self-supervised depth estimation models consist of a convolutional encoder-decoder architecture. Detailed information is lost in the deeper layers of the convolutional encoder whereby the convolutional decoder is unable to accurately

construct depth for these small detailed objects. This is a common problem in convolutional encoder-decoder architectures (27). Accurately predicting depth for small objects is important for several tasks. For example, an autonomous driving car should capture small objects further away from the car as soon as possible. Since the car could drive at high speed and must make decisions quickly based on what it observes. Or imagine a robot picking small fruits. While the robot is picking it should understand how far away the fruits are from the robot to correctly pick the fruits and not break them.
Next to the small objects, there is a second subject where the current self-supervised depth estimation models are having difficulties. The models are having difficulties correctly predicting depth for reflective/transparent objects. It turns out that these models give incorrect depth values for these objects. See figure 1.2 for an example. The reason why the models fail to correctly give depth to these objects is because there is a different depth behind these transparent objects. The model predicts this "second" depth because it is unable to understand that these transparent objects have a depth itself. This might be a problem for a robot that is walking and might walk into a glass door or is unable to clean a reflective/transparent object. To overcome these two problems mentioned above, this research focuses on self-supervised depth estimation for small objects and reflective/transparent objects. In this research, three experiments are performed to overcome these problems. The monodepth model (13) is used as a baseline model. This is one of the best self-supervised depth estimation models and other recent self-supervised depth estimation models used this model as their baseline as well (19), (4), (15). Another advantage is that monodepth can be trained with stereo pairs but also with monocular videos.
The first experiment tries to improve the depth estimation of regions with small objects by the use of attention maps from a pretrained transformer network (3). A loss function was created based on these attention maps which gives attention to objects further away from the camera to improve the depth estimation for these regions. The second experiment tries to solve the depth estimation for transparent/reflective objects. With the use of canny edge, detection (2) and the use of the attention maps from (3) this has been attempted to solve. The last experiment tries to solve the overall depth estimation by adding a transformer block to the baseline. This was inspired by the influential paper "Attention is all you need" from 2017 (27). (27) Introduced transformer models in the field of natural language processing and (31) introduced transformer models in the field of computer vision. Based on these papers the baseline architecture was updated with a transformer block and dilated convolutions in the depth encoder.
The main contribution of this research consist of the following:

- We come up with a loss function that gives more loss to the regions containing small objects by the use of attention maps from a pretrained transformer model (3). In this way, the model should learn to accurately predict depth for smaller objects.

- A second loss function was designed that gives more loss to reflective/transparent objects by the use of attention maps from a pretrained transformer model (3). This

pushed the model to give reflective/transparent objects the depth of the object itself instead of the objects behind it.

- By the use of attention maps from a pretrained transformer model, semantic awareness is introduced in the field of self-supervised depth estimation. The semantic awareness should help the model to improve depth estimation in specific regions.
- The monodepth architecture (the depth encoder) is updated with the addition of a transformer block and dilated convolutions. Since a lot of detailed information is lost in the deeper layers of the encoder, these layers might overcome this. These specific layers have a higher receptive field compared to standard 2D convolutions.

Figure 1.1: Example where scenes (biker, trees, houses) further away from the camera are ignored in the depth estimation

Figure 1.2: Examples where depth gaps appear in reflective/transparent objects

Chapter 2

Background

In this section we give all the background information needed to understand the experiments and results. First, the general building blocks of depth estimation will be explained. Then, we will clarify the baseline model (monodepth) in depth. How the model learns to predict depth, what makes this model unique, and its properties. Thereafter we explain the detection transformer model (DETR) which we will use in the experiments.

2.1 Depth general

2.1.1 Camera intrinsics

One of the building blocks for depth estimation is converting a point on an image to a point in 3D space. To understand how to convert a point on an image plane in 2D (u, v) into a 3D point (X, Y, Z) the camera intrinsics must be known. This is an essential part of self-supervised depth estimation since the camera intrinsics must be known for that approach. To convert a 2D point to a 3D point in space, the following equation must be solved:

$$S \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} \ Rt \ \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.1}$$

S is the scaling factor and is most of the time 1. U and v represent the point on the image plane. K is the matrix with the camera intrinsics (explanation will follow). R and t are the rotation and translation of the camera to the world frame. This is the position of the camera (translation) and orientation (rotation) of the camera in the real world. These are also called the camera extrinsics. X, y, and z represent the point in 3D space which is the goal. Once the above equation is solved 2D points from an image can be converted to points in 3D space. Therefore the camera intrinsics matrix K must be known. See the parameters of the K matrix below.

$$K = \begin{bmatrix} f * m_x & y & u_0 & 0 \\ 0 & f * m_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.2}$$

F is the focal length. This is the angle of view of how much the scene will be captured and the magnification of how large individual objects will be. For example, a short focal length means a wider angle and a lower magnification. M is the scaling factor in x and y (usually 1) and u and v is the pixel on the image. To solve the K matrix and equation, several images must be taken of a pattern, for example, a chessboard. The images must be taken from different positions and extra features from the images, for example, corners and dots. Then the equation can be solved by finding the camera intrinsics via these images and features. Via this approach 2D image points can be converted to points in 3D space. See figure 2.1 for a schematic overview of the camera intrinsics. All the self-supervised depth estimation models use this K matrix when learning to predict depth. An explanation will follow in a later section.

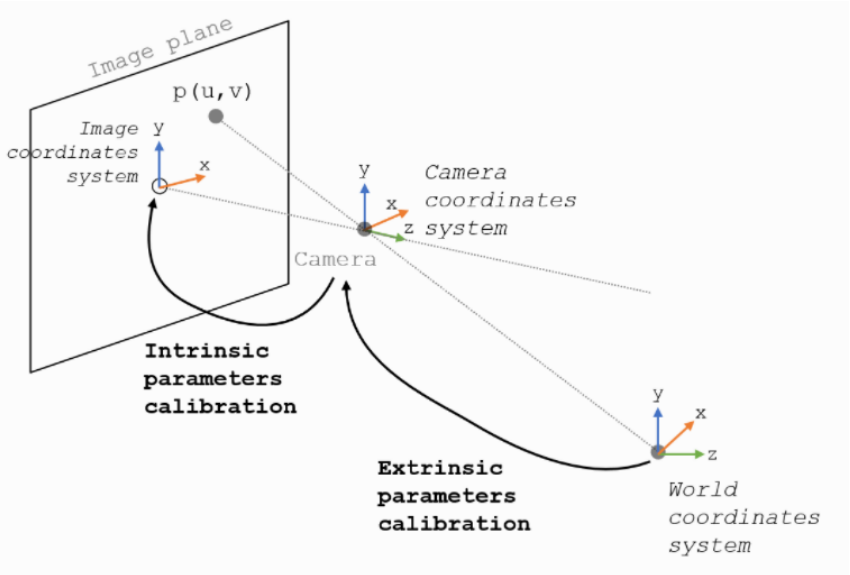

Figure 2.1: The camera intrinsics parameters

2.1.2 Stereo depth estimation

Before the introduction of deep learning and self-supervised depth estimation, it was possible to determine depth based on stereo image pairs. A stereo image pair are two images taken with two cameras, of the same scene but where there is a bit of distance between the two cameras. See figure 2.2 for a schematic overview.

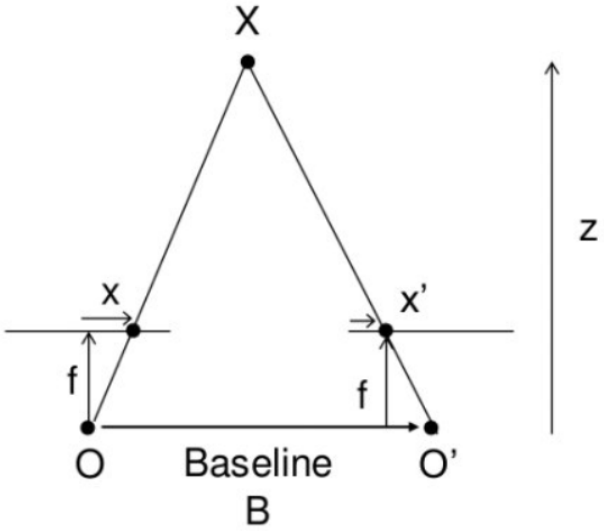

Figure 2.2: Schematic overview of the setup of creating a stereo image pair.

Based on two stereo images it is possible to get a per pixel depth image via an intuitive way following some mathematical rules. The following equation must be solved:

$$disparity = x - x' = \frac{Bf}{Z} \tag{2.3}$$

X and x' are the two points on the image plane in 2D which correspond to the real-world coordinates in 3D as explained in the section above. B is the distance between the two cameras. This is something you can measure and it is thus known beforehand. F is the focal length of the camera and is also determined beforehand. By filling in the formula a per-pixel disparity can be calculated. See figure 2.3 below for an example.

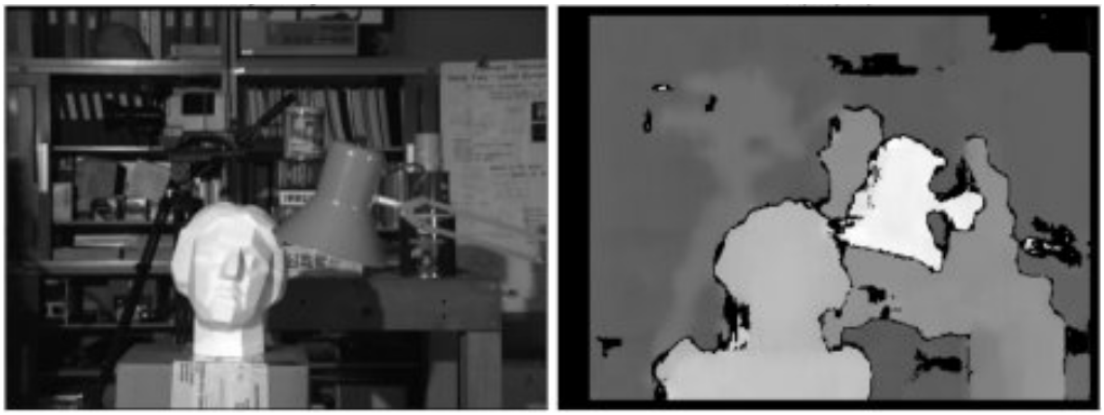

Figure 2.3: Example of a stereo image and its disparity

Unfortunately this approach contains some disadvantages as also can be seen in the

example. Some parts of the image do not contain enough texture, for these regions it is difficult to accurately calculate the depth. Imagine a scene that only contains a large, flat object which is uniform brightness and does not contain any texture. In this case, there is no information available to determine disparity (12). This can be seen in the top right corner of the example. Another problem is when occlusion occurs, so when an object is visible by one camera but is occluded in the other camera. With the arrival of deep learning these stereo pairs can be replaced by monocular videos. See the section below for more information.

2.1.3 Deep learning

Since the arrival of deep learning the depth estimation field has improved quickly. This section gives an overview of how the field changed over the last years.
First, supervised methods were introduced with the use of labels coming from sensors (16). The drawback of using these labels is that they contain noisy and missing values. Next to this drawback, the need for depth estimation in different environments arose. When a supervised model was trained in an agriculture environment it was not able to perform well in urban cities or indoor environments (for example). Since labelling data is expensive and time consuming the introduction of self-supervised monocular depth estimation came (33), (13), (28), (19), (4). It results that these self-supervised methods perform better than most supervised methods. This means that predicting depth is now possible without labels. Training is done by using stereo pairs (13), monocular videos (33), (13), (4), (19), (28) or both (13). However, stereo pairs are not nearly as widely available as monocular videos. Another advantage of monocular videos is that the model itself will learn ego-motion (4) which means that the model will learn rotation and translation parameters. The approach of self-supervised depth estimation using monocular videos will be explained in the following paragraph.

2.2 Self-supervised depth estimation using monocular videos

2.2.1 Approach

In this section, we explain self-supervised depth estimation using monocular videos. This means that there are no labels used during training. The model should learn to predict depth from images it receives from the dataset. The approach is as follows, the model receives three consecutive frames from a video where there is a small difference in time between the three frames. The middle frame is called the target frame (I_t), the other two frames (before and after the target frame) are called the source frames (I'_t). The goal of the network is to reconstruct the target frame ($I_{t'} > t$) via neural network predictions. Note that the neural networks are not predicting the reconstructed target image but a depth estimation and pose parameters. Via this, the reconstructed target image can be calculated via some mathematical rules (explanation will follow). Since there are two source images, two reconstructed target images will be calculated (one per source image).

The loss function measures how similar the reconstructed target images are compared to the target image. It calculates a per pixel photometric reconstruction error, see formula 2.4 below.

$$L_p = \sum_{I_s} pe(I_t, I_{t'} > t) \tag{2.4}$$

A summation is taken over the two reconstructed target images, and the photometric reconstruction error (pe) is calculated between the reconstructed target images and the target image. See formula 2.5 for the photometric reconstruction error.

$$pe(I_a, I_b) = \frac{a}{2}(1 - SSIM(I_a, I_b)) + (1 - a)||I_a - I_b||(a = 0.85) \tag{2.5}$$

The photometric reconstruction error measures the similarity between two images. The first term is the structural similarity index measure (SSIM). The difference between other techniques (mean squared error, peak signal-to-noise ratio) and SSIM is that these methods calculate the absolute pixel difference while SSIM calculates similarities within windows of an image. The second term of the equation calculates the absolute difference per pixel. When the pe loss is zero it means that (one of) the reconstructed image(s) is identical to the target image. If the pe loss is high it means that the reconstructed image(s) will more look like one of the source images (explanation will follow). Both the neural networks (depth and pose) learn via the above loss function.
For the network to correctly reconstruct the target frame it should learn two different tasks simultaneously. Namely, the depth estimation of the target frame and the relative pose between the source and target frames. At first, the pose parameters are explained. The target and source frames are fed to a pose encoder network (3 images in total). This network converts the input images into features. These features are sent to the pose decoder which predicts the relative pose (rotation and translation) between the source frames and the target frame. Per source-target image pair (two pairs in total) it predicts 6 parameters (3 translation and 3 rotation). These are called the 6 degrees of freedom (6-Dof). These parameters refer to the ability of a rigid body to move in three-dimensional space. The 6-Dof explains the ability to move forward/backward (surge), up/down (heave), and left/right (sway), also called translation. They also explain the rotation of the three perpendicular axes. It is thus a transformation matrix that contains the 3 rotation and 3 translation parameters between a source and target frame. In this paper it will be noted $T_t > t'$. See figure 2.4 for a visual example of the 6-Dof.

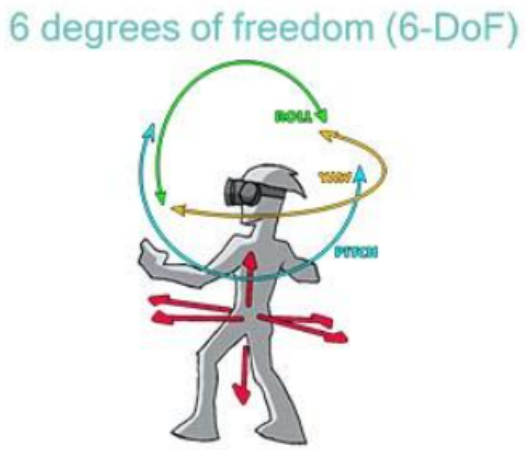

Figure 2.4: A rigid body in three dimensional space. It is able to perform translation (forward/backward, up/down and left right) and rotation in the three perpendicular axes.

Next to the pose network a separate network is trained simultaneously to predict depth. The target image is sent to the depth encoder. This encoder converts the target image into features. These features are converted to a depth image in the depth decoder (D_t). With the outputs from the neural networks (depth image and pose parameters), the reconstructed target image can be calculated. See figure 2.5 for an schematic overview.

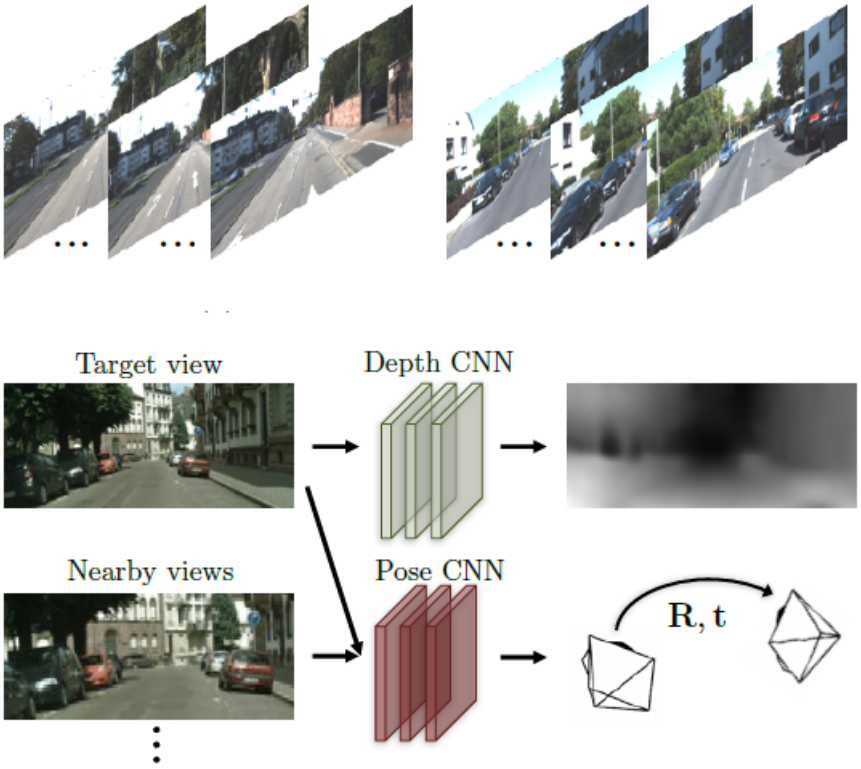

Figure 2.5: During training 3 consecutive images are sampled. The target image is sent to the depth network to estimate the depth image. The target and source images are sent to the pose network for pose parameter estimation. Both the neural network predictions are used for calculating the reconstructed target image.

The depth estimation together with the pose parameters prediction is used to reconstruct the target image. In addition, a camera intrinsic matrix (K matrix) is used. This contains the intrinsic parameters of the specific camera model, namely the focal length, image sensor format, and principal point. Most data sets contain this information, this is also the case for our dataset, Kitti (20). The idea is that pixels from the target image I_t are projected onto the source image I'_t based on the depth estimation of the target image D_t, the pose parameters between the target and source image $T_t > t'$ and the K matrix. Intuitively a pixel is sampled from the target image and the position of this pixel is changed based on the neural network predictions. Some examples will follow to indicate how the predictions influence the position.
If the pose parameters are high because there is a lot of movement between the target and source frame, the pixel will change from a position based on the pose parameters prediction from the neural network. Note that the same pose parameters are applied for every pixel. If the depth estimation for a pixel is small this means that the pixel is close to the camera and thus the pixel will change a lot. If the depth is big for that pixel the

change will be relatively small since that pixel is further away in the scene. See formula 2.6 for the equation on how to reconstruct the target image $I_{t'} > t$

$$I_{t'} > t = I_{t'}\Big\langle proj(D_t, T_t > t', K)\Big\rangle \tag{2.6}$$

$$\langle\rangle = Samplingoperator$$

Proj is the resulting 2D coordinates of the projected target image pixels onto the source images I'_t. The target pixels are thus projected onto the source image based on the depth estimation, pose estimation, and K matrix. Since there are two source images (each with its pose parameters) there are two reconstructed images calculated. Notice that the projected coordinates are continuous values but pixel values have an absolute position. Therefore bilinear sampling is used afterward to give every pixel an absolute position. The distance between the 4 surrounding pixel positions is calculated. The pixel position with the shortest distance is the end position of the projected coordinate. See figure 2.6 for an illustration.

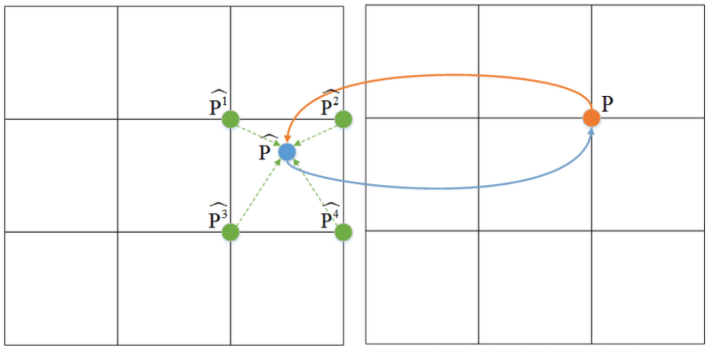

Figure 2.6: Use bilinear sampling to map the position of a projected coordinate to a absolute pixel place

After training there is a trained network that can predict depth based on single images. To get depth images the depth network gets extracted from the other parts. After training color images are fed to the depth network and depth images will come out. This depth network can then be used for several tasks. Note that the pose network is during inference not needed, it is only used for reconstructing the target image and this is only necessary during training. After training only the depth network is necessary for converting color images to depth images.

2.2.2 Monodepth model

In this section, we will explain the baseline monodepth model. First, we motivate why monodepth has been chosen as the baseline model. Then we cover the the model architecture and the models trick to overcome occlusion.

In this research we use the monodepth model as baseline (13). This model is one of the most recent self-supervised depth estimation models with state-of-the-art results. Other recent self-supervised depth estimation models use monodepth as their baseline as well (4), (19). Monodepth can be trained with stereo pairs, monocular videos, or both which makes it easy to use in practice.

Architecture

Training the monodepth model is the same as other self-supervised depth estimation models as explained in the approach section above. Monodepth trains with predicting depth images at different scales. This reduces artifacts and improves the quality. Like other self-supervised depth estimation models, monodepth has a depth network and a pose network. See figure 2.7 for a schematic overview of the monodepth network architectures. First, we look at the depth network. On the left, the depth network architecture can be seen. As input, the target image is received. The image is encoded using 2D convolutions. The kernel size is 3 for all the layers except for the first 4 convolutional layers with 7; 7; 5; 5, respectively. The number of output channels for the first convolutional layer is 32. The input image is encoded with different amounts of channels, width, and height. These encoded feature maps are encoded at 4 different scales indicated by the green arrows. All convolutional layers are followed by a ReLU activation except for the prediction layers, where $\frac{1}{(\alpha * sigmoid(x))+\beta}$ where α and β are chosen to constrain the predictions between 0.1 and 100 units. The 4 depth images are upscaled using bilinear sampling afterward such that the reconstructed target image has the same width and height as the target image. Next, the pose network is explained. The pose network receives 3 images, the target image, and two source images. The images are concatenated along the color channels. These images are encoded using 2D convolutions. The number of output channels for the first convolutional layer is 16, and the kernel size is 3 for all the layers except for the first two convolutional and the last two deconvolutional layers where 7; 5; 5; 7, was used respectively. The output of the network is 6-Dof per source-target image pair. All convolutional layers are followed by a ReLU nonlinearity except for the last layer where no nonlinear activation is applied.

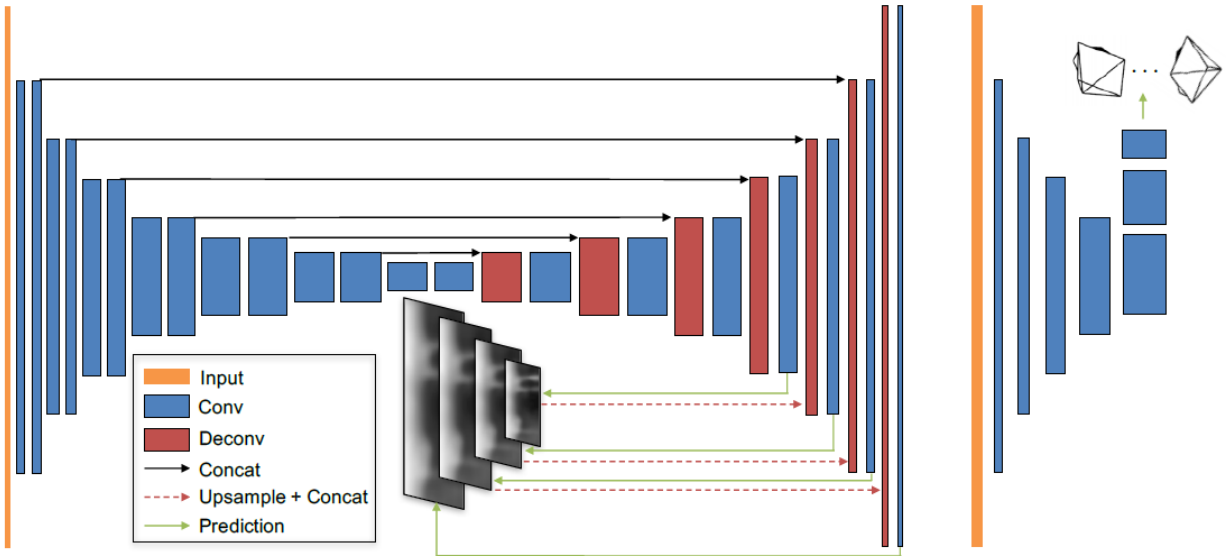

Figure 2.7: Network architecture for depth (left) and pose (right). The width and height of the blocks indicate the output channels and the spatial dimension of the feature maps at each layer. Each reductions / increase of a block indicates a change by the factor of 2. Image coming from (33).

Per-pixel minimum reprojection error

As mentioned before there are two source images, so two reconstructed target images are calculated. Most self-supervised monocular depth estimation methods average the reprojection loss over the two reconstructed images and the target image. This method could create issues with pixels that are visible in the target image but are not visible in one of the source images. Because if some pixels from the target image are not in the source image(s), these pixels can not be reconstructed (see example in figure 2.10 above). Imagine that the depth network correctly predicts a depth pixel for such a scenario. Then the corresponding color in the occluded source image will likely not match with the target image and will therefore receive a high penalty. These problematic pixels can occur because of two reasons: out-of-view pixels due to the motion between frames and occlusion. Instead of taking an average loss over the two reconstructed images, minimizing the loss over the reconstructed images is performed. The loss from equation 1 is slightly changed, see equation 2.4. This means that per pixel the loss is calculated between the two reconstructed images and the target image. The pixel with the lowest loss is used. See figure 2.8 for an overview.

$$L_p = \min_{t'} pe(I_t, I_{t'} > t) \tag{2.7}$$

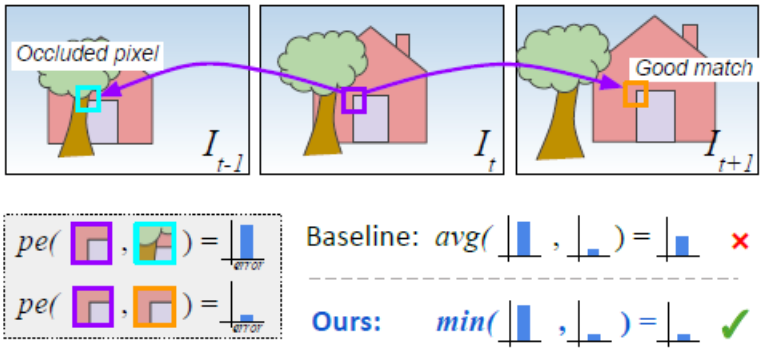

Figure 2.8: Example where a specific pixel is visible in the target image (I_t) but only in one of the source images (I_{t-1}, I_{t+1}). Both source images reconstruct the target image. Instead of averaging the loss over those two reconstructed images, a minimum per pixel loss is calculated over the two reconstructed images, leading to sharper results.

2.2.3 Behaviour during training

To give an understanding of how the model behaves at the beginning and end of the training, the reconstructed image and network predictions are analyzed. See figure 2.9 for an example at the beginning of training.

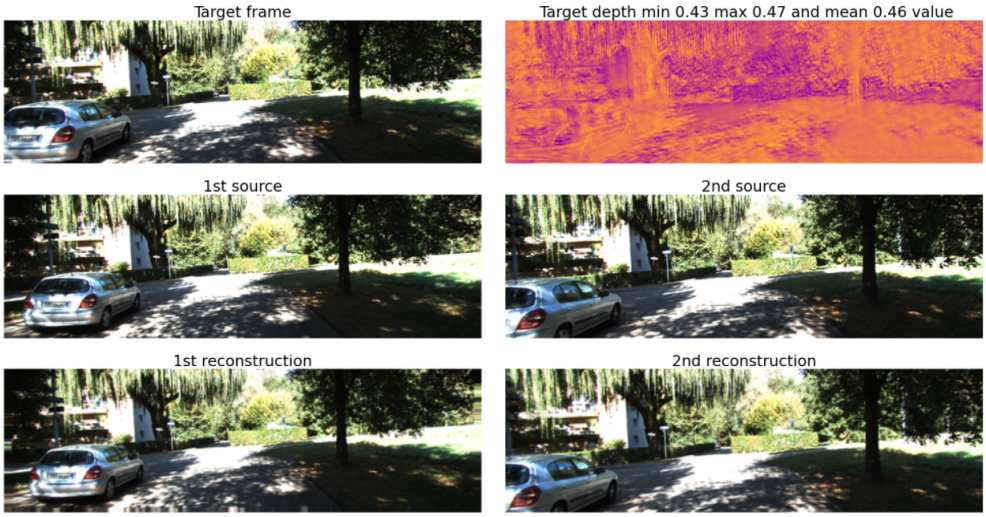

Figure 2.9: Reconstructing the target image at the beginning of training with corresponding depth estimation

Since there are two source images, two reconstructed target images are calculated. As can be seen, the reconstructed images are copies of the source images at the beginning of training. The reason for this behavior is that the depth image is almost uniform for every pixel. This means that the model does not know how to predict depth yet and

gives every pixel roughly the same value.
Given this result at the beginning of training, it is interesting to see how the model creates the reconstructed images at the end of the training, see figure 2.10 for an example.

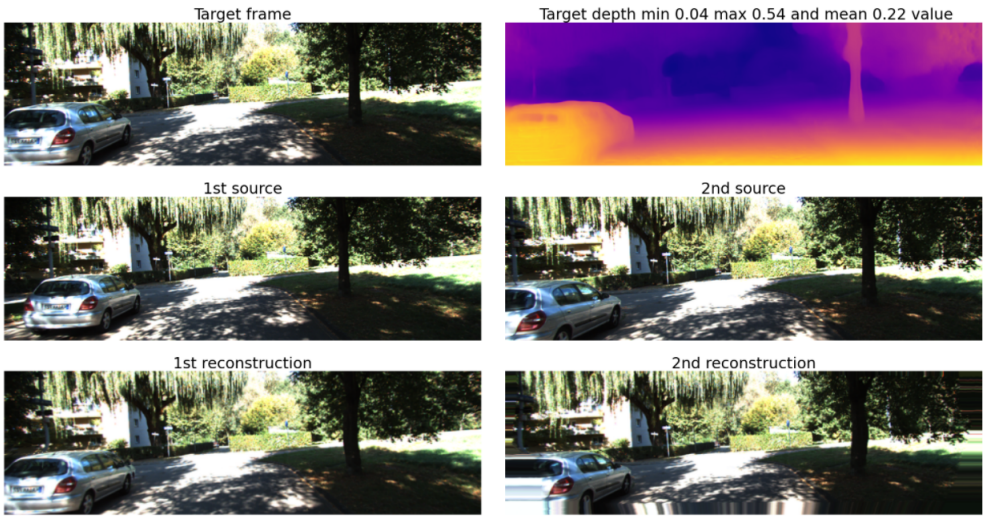

Figure 2.10: Reconstructing the target image at the end of training

Looking at the reconstructed target images in figure 2.10 it is visible that there is a clear difference in the source and reconstructed images. Compared to the reconstructed images at the beginning of training from figure 2.9, the reconstructed images are almost copies of the target frame. Looking at the second reconstructed image it can be seen that the model is having problems at the left bottom corner. This can be explained by the fact that the model should reconstruct a part of the car which is not visible on the second source image but only in the target frame. This is a problem in self-supervised depth estimation. The authors from monodepth (13) came up with a solution to overcome this problem. See the section per-pixel minimum reprojection error above for the proposed solution.

2.3 Transformer Models

This research makes use of a transformer network in the form of a detection transformer model (DETR). In this section we give the motivation of using a transformer network and why DETR is chosen (advantages and disadvantages). The section also covers the training approach and its architecture and some attention map examples will be given. At last, we explanation how a transformer block works in detail. Next to the attention masks from DETR, a transformer block will be used in one of the experiments.

Since the arrival of the influential paper "Attention is all you need" in 2017, transformer models were introduced and used (27). First, this type of model was applied in the field

of natural language processing (NLP). Since transformer models have a better long-range dependency between words compared to the standard recurrent neural networks (RNN) (9) or long short-term models (LSTM) (14). RNN and LSTM models can only process words in the local word neighborhood but for many NLP tasks like machine translation, a long-range dependency is desirable (27). In 2019 self-attention was introduced in the field of Computer Vision (CV). Zhang introduced self-attention during training a generative adversarial network (31). Self attention has also been proposed for image classification (1), object detection (21) and semantic segmentation (30). All these tasks have improved results compared to convolutional layers (5), (7), (32) because convolutional layers can only handle spatial information from local pixel neighbourhood while transformer layers have access to the complete image at once because of the global receptive field (22). See figure 2.11 below for the difference of the receptive field between a convolution and a transformer. Note that a transformer is sometimes also called attention since the transformer pays attention to certain regions.

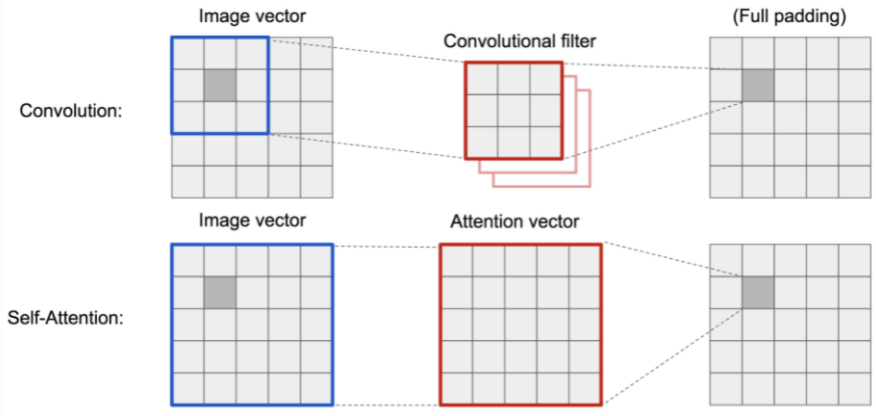

Figure 2.11: Convolution operation and transformer have access to image regions of very different sizes.

2.3.1 Approach

DETR (3) is an object detection model which forces unique predictions via bipartite matching. Given a fixed small set of learned object queries, DETR is reasoning about the relations of the objects and the global image context. The model is trained on the COCO dataset (18) which contains over 300K labeled indoor and outdoor images including 80 different categories. This makes the model generalize able to perform well in different environments. Training of the DETR model on a specific dataset is thus not necessary. This is one of the reasons why we use DETR in this research.

The training pipeline of the DETR model is as follows. The model receives an image from the COCO dataset. First, this image is converted to features using convolutional layers. These features are sent to the transformer encoder together with a positional encoding.

The transformer decoder receives as input a small fixed number of learned positional embeddings which are called object queries. Each output embedding of the decoder is passed through a shared feed-forward network that predicts either a detection (class and bounding box) or no class. So for example, the prediction might be class "bird" and a bounding box around a bird. See figure 2.12 below for an schematic overview of the DETR architecture.

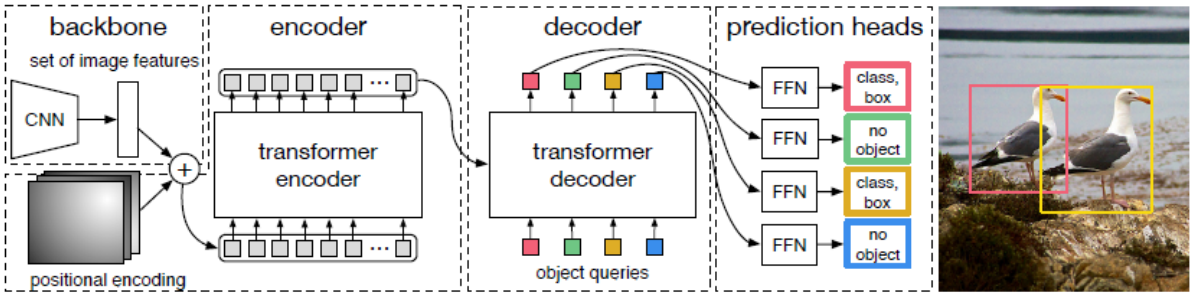

Figure 2.12: Overview of the architecture of the detection transformer model. Image coming from (3).

Next to the prediction, the transformer blocks generate attention masks to support the predictions. These attention masks will be used in this research. For every image, DETR receives it is able to produce 100 different attention masks where each attention mask gives attention to a different object. Every attention mask comes with a probability of how secure the DETR model is that it correctly gives attention to an individual object. Second, next to the probability per attention mask, also every pixel inside the attention mask has a probability. See figure 2.13 for some example attention maps coming from the model. Note that each attention map is paying attention to a single object.

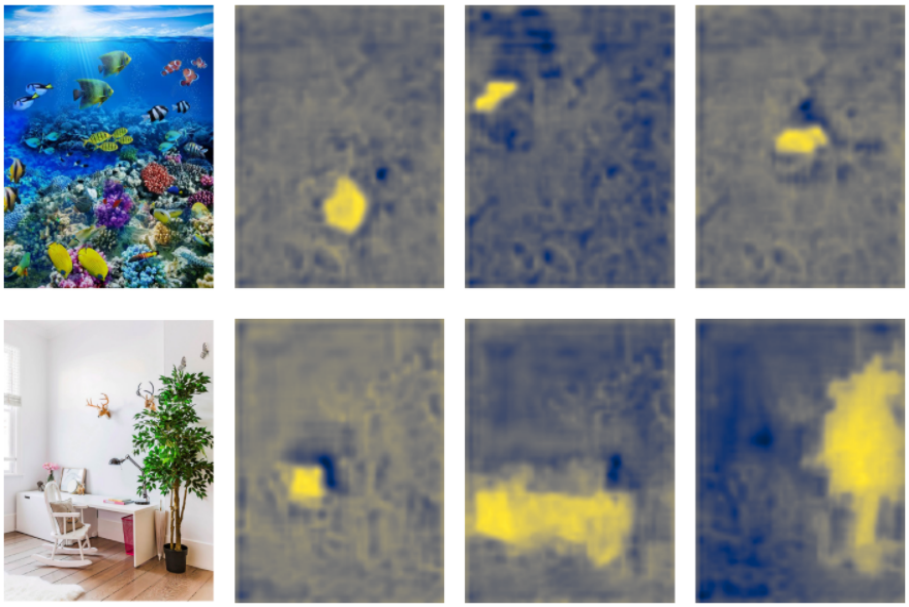

Figure 2.13: Attention maps from DETR in different environments

A advantage of the attention masks is that not only the attention mask itself has a probability but also every pixel within the attention map. In this way, it is possible to filter out pixels where the model is less confident off. A disadvantage of the model is that it always produces 100 attention masks based on 1 image independent of the number of objects in an image. Imagine an image where there are almost no objects like the sky, meadow, or lake. The model will then still produce 100 attention masks. Then, for some objects, there will be some overlap between the attention masks. Another disadvantage of large attention maps is that they can capture multiple objects. For example, if a DETR model tries to give attention to a car road where there are some poles on the road, the model is not always good at filtering out these poles. Or if it tries to give attention to all the trees in the background of some view and there are some objects in the foreground, the model also gives some probability to the objects in the foreground.
The attention masks from the DETR model might help improve the monodepth model. As mentioned before in the introduction, one of the drawbacks of the monodepth model is that it is not focusing on objects further away from the camera (see figure 4.8). Monodepth is also having difficulties in correctly estimating depth for reflective/transparent objects (see figure 4.4). For both drawbacks, the attention masks from the DETR model will be used in an experiment. The attention masks are able to give attention to individual objects in the scene, in this way the monodepth model can benefit from this additional spatial information.

2.3.2 Transformer block

At last, we explain in detail how a transformer block works. This transformer block will be used in one of the experiments to learn to generate attention masks while predicting depth. We use the dilated convolutions in the same experiment and introduced it in the related work, in the next section. We explain in detail how the transformer block is used with dilated convolutions in the method section. First, we explain how a transformer block works.

Self-attention can give attention to multiple regions in an image. This might be beneficial for dense predictions since multiple regions in an image can benefit from the same attention. For example, two cars that are driving at different sides of the road but have an equal distance to the camera. The cars might receive attention from the transformer block but would not be captured by the same kernel in convolutions since there is too much distance between the cars. In 2019 (31) used self-attention during training a generative adversarial network. In figure 2.14 some examples can be found from this research. It is visible that the attention maps try to give attention to similar regions in the image based on the query point. The attention map can give attention to different regions in the image.

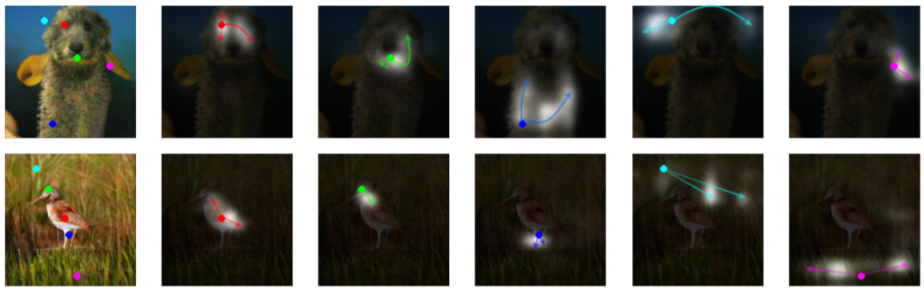

Figure 2.14: Example of attention maps. The dots on the left image represent query points. The other images on the same row indicate the self attention maps based on the query points. Attention maps coming from (31).

The attention maps are calculated as follows. First, the convolution feature maps (channels x width x height) are received. Then 1x1 convolutions are applied on the feature maps such that the key f(x), value h(x), and query g(x) values are computed. Key and query learn to sample important regions. Value learns a representation of the received feature maps. The 1x1 convolution is a trick to lower the number of channels but keep the spatial structure (width and height). A softmax is calculated over the dot product between the key and query to calculate the attention map a_{ij}.

$$a_{ij} = softmax(f(x_i)^T g(x_j)) \tag{2.8}$$

The attention map is a tensor of values between 0 and 1 to indicate which regions should receive attention and which not. Then the attention map is multiplied by the value such

that the self-attention feature maps are created.

$$o_j = W_v(\sum_{i=0}^{N} a_{ij} h(x_i)) \tag{2.9}$$

o_j has the shape of channels x width x height. Where each channel is a different attention maps which gives attention to some regions in the image. See figure 2.15 for an schematic overview.

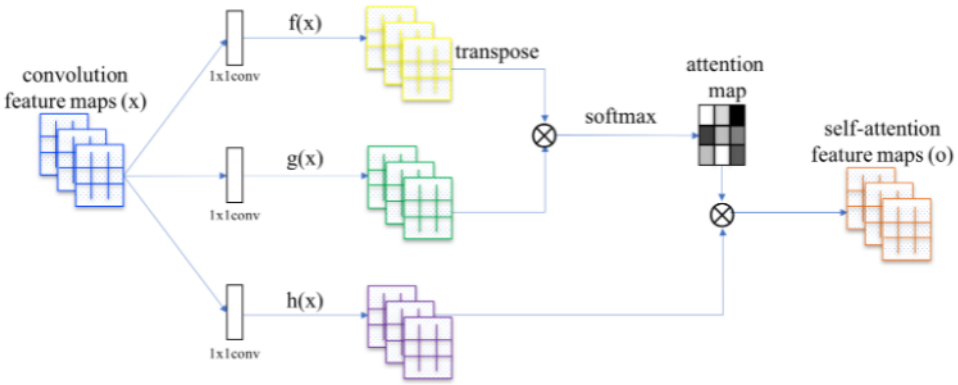

Figure 2.15: Overview of the self-attention block calculation

Chapter 3

Related work

In this section we give an overview of relevant recent related work. First we explain other self-supervised depth estimation methods which are variations of the monodepth model. Next to that we give an overview of transformer networks in the field of depth estimation and why it might help to improve monodepth. At last we explain dilated convolutions which we will use in one of the experiments as well.

3.1 Depth estimation

Since the arrival of monodepth (13) in the depth estimation field, several other self-supervised depth estimation models came available. The general goal of self-supervised depth estimation is to synthesize the appearance of the scene seen from a novel camera viewpoint. Most methods in self-supervised depth estimation use the photometric reconstruction error (33), (13), (19), (28), (4) (monodepth as well). The goal is to reconstruct a target frame by using a source frame, camera movement, and depth estimation. Along this photometric reprojection loss in the self-supervised method other additional loss functions are proposed: structural similarity index measure (SSIM) (13), (19), (28), (4) which is less sensitive to absolute pixel to pixel comparison but compares windows of images, iterative closest point (ICP) (19) which improves the camera movement estimation and edge awareness (13), (19). Some training artifacts like infinite depth (black holes) and occlusion / out of view pixels are studied. (13) Proposed masking pixels where infinite-depth occurs. This decreases the loss during training but does not prevent infinite-depth during inference. (4) Corrected the infinite depth by using object height priors and (13) proposed a slightly different photometric reconstruction error to skip the occluded / out of view pixels. (19) Used a principled mask to mask pixels that are unexplainable due to any reason and will therefore not influence training. The drawback of most self-supervised methods is that they assume a static world (33), (13), (19). Therefore depth estimation of moving objects is not well learned. Recently some successful methods are published which can handle a dynamic world with either using optical flow (28), geometric structure (4) or regularizations (17).

3.2 Transformer networks

Recently, transformer models are used in the field of depth estimation. (22) Used a transformer architecture combined with different datasets for supervised depth estimation and image segmentation which resulted in state-of-the-art results. They used transformer layers to improve the overall depth estimation. (15) Used a combination of convolutional layers and a transformer layer in self-supervised depth estimation to also improve the overall depth estimation. Transformer networks have a promising future compared to convolution networks but they have more parameters and thus need more data for good results (22).
But so far, there has not been any research in self-supervised depth estimation for small objects and reflective/transparent objects. Self-supervised depth estimation models are having difficulties in correctly predicting depth for these objects as explained in the introduction. With the arrival of transformer models in the field of computer vision it is possible to pay attention to specific objects (3). Since transformers have shown their potential in the field of computer vision and depth estimation and are able to give attention to single objects they might improve the depth estimation for small and reflective objects. A pretrained detection transformer (DETR) is the model we will use in this research (3). This model is trained with objects detection and their attention maps are thus focused on single, unique objects. The model is trained on the COCO dataset and is thus able to generalize well to different environments. See figure 2.13 for some examples of attention maps in different environments where the models pays attention to single objects. In the method section, a complete explanation will be given of how the attention maps from DETR will be used to improve the depth estimation for small and reflective/transparent objects.

3.3 Dilated convolutions

Dilated convolutions were introduced in 2017 in the field of semantic segmentation (6). Since then various ResNet architectures have been update with dilated convolutions (32), (29) and (8). Dilated convolutions can capture a bigger receptive field compared to regular convolutions. This is because there is space between the kernel values based on the dilation rate. Therefore dilated convolutions are sometimes also called "hole algorithm". The bigger the dilation rate the bigger the receptive field. See figure 3.1 for an example with different receptive fields.

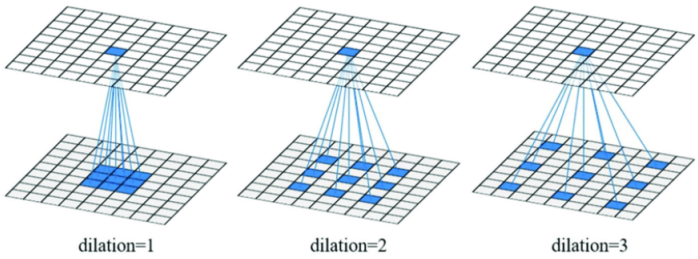

Figure 3.1: Example of dilated convolutions with different dilation rate numbers

For dense predictions (segmentation and depth) the model can benefit from a bigger receptive field since the pixels in the image are correlated. This is different for sparse predictions such as object classification where only a part of the image is relevant (the object). There has not been a lot of research done in applying dilated convolutions with transformer blocks in the field of self-supervised depth estimation. In one of the experiments dilated convolutions will be used together with a transformer block. For the full explanation see the transformer block subsection in the method section. The expectation is that self-supervised depth estimation can benefit from dilated convolutions.

Chapter 4

Method

In this section, we propose and explain three experiments. At first, we explain the Kitti dataset which we will use in the experiments. The experiments will be compared with the monodepth model (baseline) which was also trained with the Kitti dataset (13). The monodepth model was explained in the background section 2.2.1. In the first experiment, the depth gaps of reflective/transparent objects are addressed by an additional loss function. The second experiment tries to improve the depth estimation of small objects with an additional loss function. The last experiment tries to improve the overall depth estimation by using a transformer block and dilated convolutions. The experiments are described in three separate sections. Per experiment, the motivation, goal, and approach will be clarified. Before the methods of the experiment will be explained the prepossessing steps of the attention masks from the DETR model will be clarified.

For the edge loss and weight mask experiment (explanation will follow) the attention masks are used which are coming from the pretrained transformer-based object detection model (DETR) (3). This model was described in the background section 2.3. For both experiments, the following prepossessing steps are performed. The DETR model generates by default 100 attention masks per Kitti image where every attention mask has a probability of how secure the model is that it correctly gives attention to an object. attention masks with a probability lower than 0.5 are removed. This seems to be the correct threshold for having a diverse set of attention masks and qualitative good attention masks. See figure 4.1 below for a Kitti image with corresponding attention masks with a high and a low probability. It is visible that attention masks with a low probability (0.2) do not correctly give attention to a complete object as only a part of the road is covered for example. When DETR gives a high probability (0.9) it is visible that the model correctly gives attention to individual objects. The probability of 0.5 is chosen such that we have enough attention masks which are still correctly giving attention to objects.

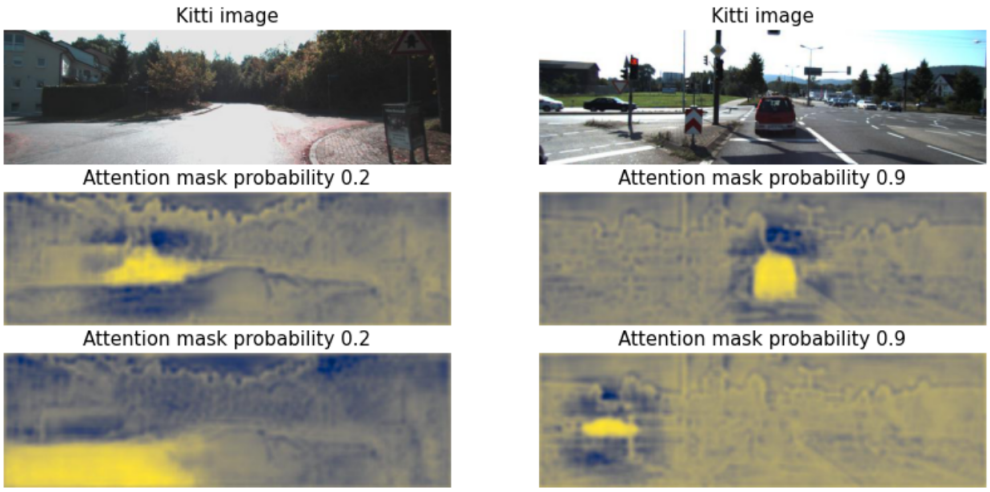

Figure 4.1: Example of attention masks from the DETR model corresponding to a Kitti image. DETR gives every attention mask a probability of how secure the model is it correctly gives attention to an object.

The second prepossessing step is converting the remaining attention masks to attention maps. The attention mask itself has a probability of how secure the DETR model is it correctly gives attention to an object. Next to this probability, the attention mask has also a per-pixel probability. High probability indicates attention to an object, low probability indicates no attention. The attention mask pixels which have a probability lower than 80% are mapped to 0. This number is 5% lower than the recommend percentage by (3). The attention mask pixels which have a probability higher or equal to 80% are mapped to 1. After the conversion, the attention mask will be called attention map in this research. The reason for this update is to fully give attention to a specific object which might help in the experiments. Note the difference between an attention mask and an attention map. See figure 4.2 for an example.

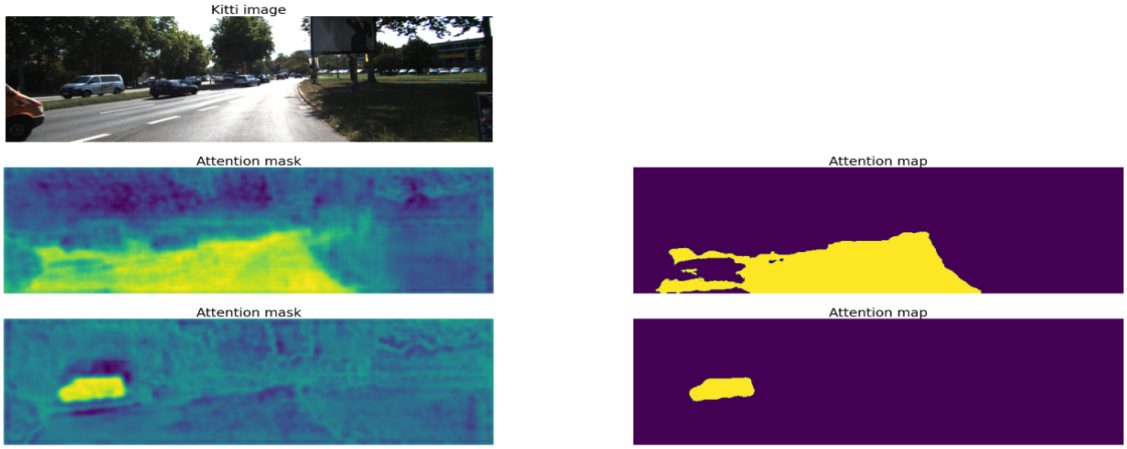

Figure 4.2: Kitti image with two corresponding attention masks. Pixel values ≥ 0.8 are casted to 1, pixel values < 0.8 are casted to 0. Resulting in the attention map.

4.1 Dataset

The Kitti dataset is the dataset we are using in this research in all experiments. Almost all self-supervised methods are trained on the KITTI dataset (33), (13), (19), (28), (4) where videos are taken during driving a car in Germany on highways, cities, and neighborhoods. Our baseline model, monodepth is also trained with the Kitti dataset (13). Kitti is a diverse dataset of different environments. The dataset contains stereo and monocular frames. The dataset contains over 40.000 images. The advantage of Kitti is that it contains around 700 labeled images coming from a sensor which can be used for evaluation and comparing models. A disadvantage is that these values are sparse and contain a lot of zero values. See figure 4.3 for an example. However, this dataset is used in almost all the self-supervised depth estimation papers and makes it easy to compare with. In this paper, the pre-processing steps from Zhou et al.'s (33) and (13) are followed to remove static frames. This results in 39.810 monocular images for training and 4.424 for validation (same as baseline model).

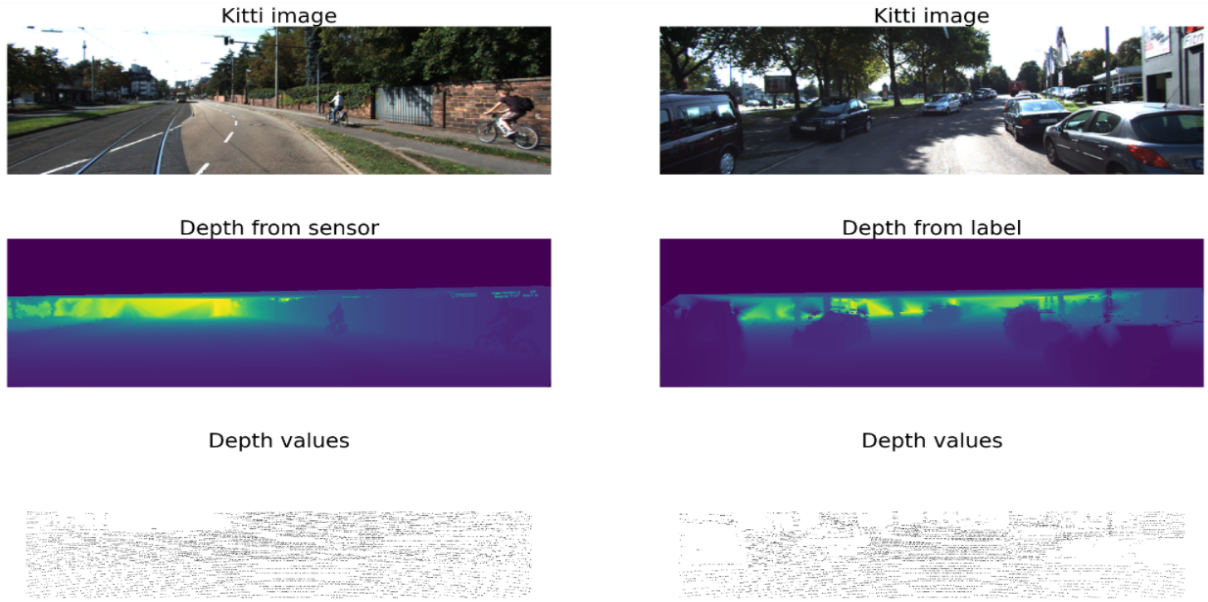

Figure 4.3: Top row Kitti image. Middle row the corresponding depth label. Bottom row the same depth label where the dots indicate a depth value and the gaps in between indicate missing values. Note the missing values in the top part of the image.

4.2 Edge loss

In this section, we explain the first. The goal of this experiment is to reduce the depth gaps of reflective/transparent objects as explained in the introduction.

A common problem in the monodepth model is that it predicts relatively big different depth values for transparent/reflective objects. For simplicity, in this research, this will be called gaps. In this experiment, an additional loss term is introduced to reduce these depth gaps. When the lambertian property is violated these gaps might appear. The lambertian property states that the surface of an object must remain the same if the observer changes its viewpoint. But when an object is reflective or transparent (glass), this does not hold when the view from the observer changes. See figure 4.4 for some examples. A consequence of this is that there are large image gradients in the depth image around the borders of these gaps.

Figure 4.4: Examples where depth gaps appear when the lambertian property is violated

A possible solution is to use Canny edge detection (2) to detect edges around the gaps. Canny edge detection takes both the direction and the magnitude of the gradients into account. Canny edge detection finds edges by first calculate the first-order derivative of a Gaussian filter. Two 1D filters are applied separately, resulting in two gradients I_x, I_y in both x and y-direction. To determine the magnitude of the gradients the following formula is applied: $\sqrt{I_x^2 + I_y^2}$. Only gradients higher than a threshold are used. See figure 4.5 for an example where canny edge detection finds edges based on a depth image.

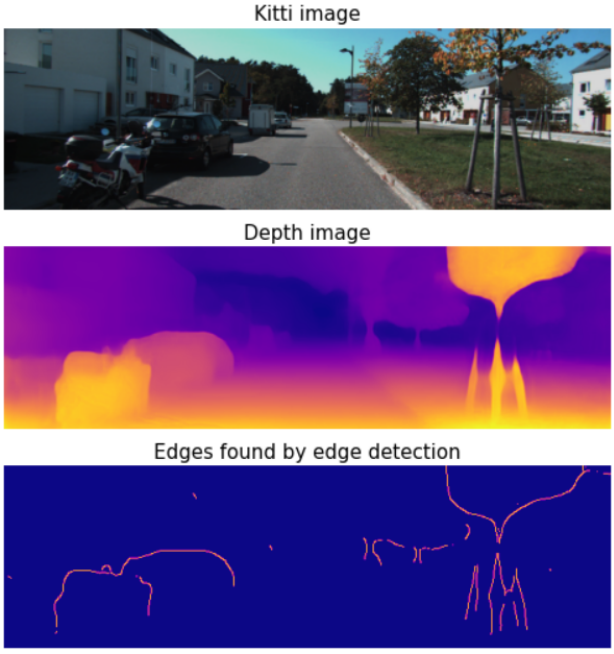

Figure 4.5: Example where canny edge detection finds edges based on a depth image

The attention masks from the DETR model are used in this edge experiment. Most of the attention masks capture individual objects and therefore they can be used to determine if depth gaps appear inside the depth estimation of these objects. The goal is to reduce

the depth gaps of reflective/transparent objects. The disadvantage of this approach is that some attention masks cover multiple objects, see figure 4.6 for an example. The loss function will probably find other edges besides the depth gaps due to these big attention masks.

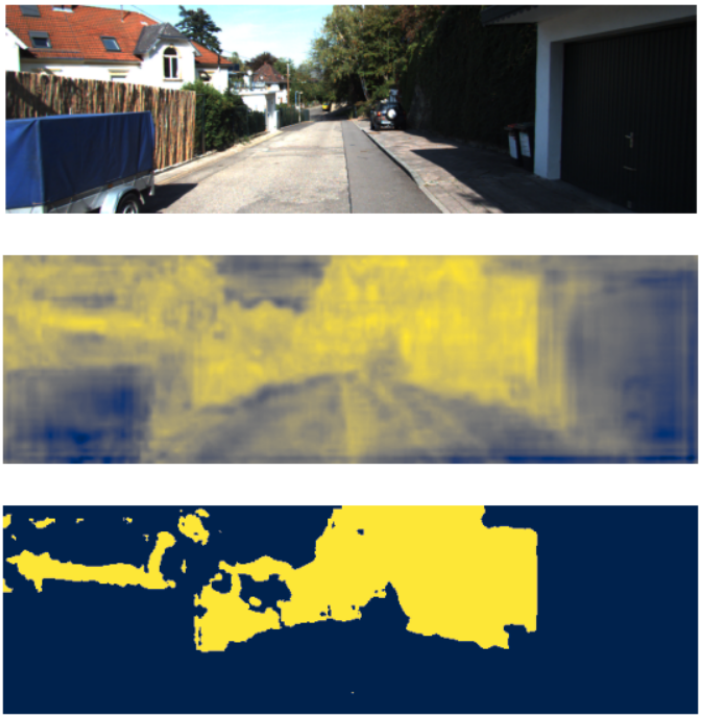

Figure 4.6: Example where an attention mask gives incorrectly attention to multiple objects. Top: Kitti image, middle: attention mask, bottom: attention map.

The attention masks from the DETR model are filtered first as described at the beginning of the method section. Attention masks with a probability lower than 0.5 are removed and the remaining attention masks are converted to attention maps. The remaining attention maps are used for edge detection. Since an edge could be found twice by a different attention map the following rule is applied. The remaining attention maps are sorted based on their size. The smallest attention map is used first to find edges inside the depth image of the region of the attention map. This is done by paste the attention map over the depth image, only the depth image inside the attention map will remain. In this region, edge detection will be applied. Then the following attention map is applied, etc. Only edges that have not yet been found by earlier attention maps in the depth image are used for calculating the loss. This approach might help to not use the attention maps which cover multiple objects as mention above. After applying the attention maps to find edges, the found edges are summed and are used as the additional loss term.
Attention maps cover an entire object, including the borders. Therefore also the object borders are found as edges. To overcome this problem, erosion is applied on the attention

map before finding edges in the depth image. Erosion helps to let the attention map shrink a little bit such that the attention map will not find object borders in-depth images as edges. See figure 4.7 for an example. It is visible that before the erosion is applied the loss function finds a lot of objects border edges. The goal of the loss function is to only find depth gaps inside the objects. Therefore erosion is applied and it is visible that after the erosion is applied the loss function only finds inner object edges (image bottom right of figure 4.7. This experiment hypothesizes that the depth gaps of transparent/reflective objects will be reduced but new artifacts might be introduced because some attention maps could still capture multiple objects.

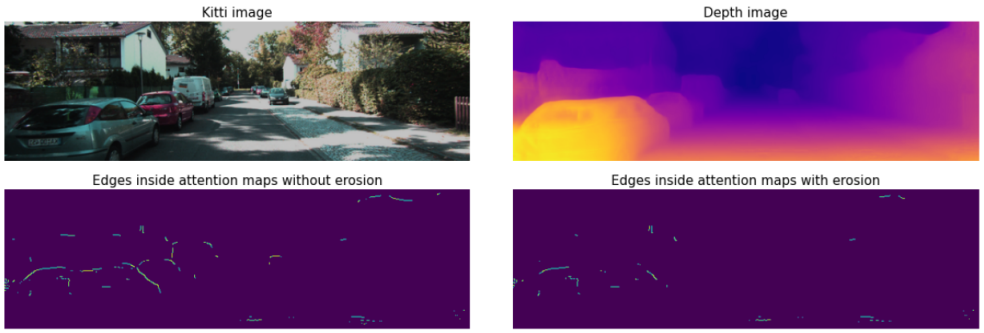

Figure 4.7: Example where edge detection is applied on the depth image based on the attention maps. The difference is shown where erosion is applied before edge detection such that object borders are not taken into account as edges. After erosion (bottom right), roughly only inner object edges are found.

4.3 Weight mask loss

In this section, we will explain the second experiment. The second experiment tries to improve the depth estimation for small objects as this is a difficult task for self-supervised depth estimation models as explained in the introduction.

The monodepth model (13) is trained with the objective that the reconstructed target image should be identical to the target image, as explained in the background section 2.2.1. Since objects which are further away from the camera contain relatively fewer pixels in the reconstructed image, the model is focusing more on the pixels close by since they contain the biggest part of the reconstructed image and therefore will help to shrink the reprojection loss the most. A downside of this approach is that objects which are relatively further away from the camera are getting less attention from the model and their depth is therefore not as accurate as objects which are close by. See figure 4.8 for an example. Another downside is that for some tasks the depth estimation is more important for some specific objects and less important for other objects. For example, a robot that is picking apples is mostly interested in the depth estimation of apples and less interested in the depth estimation of other objects in the view of the robot. In the

current monodepth model all pixels have an equal importance and it is not possible to focus more on desired regions.

Figure 4.8: Example where objects (biker, trees, houses) further away from the camera are ignored in the depth estimation

In this experiment we give more weight to pixels that are further away in the scene. This is done by using the attention maps from the DETR model (3). Monodepth has a per-pixel loss of how good the model is in reconstructing the target image. With the use of the attention maps a weight matrix is calculated (explanation will follow). This weight mask is multiplied by the per-pixel loss. Smaller objects will get more weight than larger objects since they contain a higher value in the weight mask. The hypothesis is that smaller attention maps from the DETR model give attention to objects further away in the scene and big attention maps give attention to objects close by. This idea comes from the fact that objects which are close by contain more pixels and objects further away contain relatively fewer pixels. For every Kitti image a weight mask is calculated based on the corresponding attention maps, see figure 4.9 for an example of a weight mask (explanation will follow). As explained at the beginning of the method section, the attention masks with a probability lower than 0.5 are removed. The remaining attention masks are converted to attention maps. With the remaining attention maps, a weight per attention map is determined based on their size. The smaller the size of the attention map, the higher the weight. Note that the weight of the attention maps is independent of the other attention maps for a specific Kitti image, as some Kitti images only contain large or small attention masks. The size of the attention map is divided over the average attention map size across the dataset. Then 1 divided by this number is performed to give small attention maps a high weight and big attention maps a low weight. Since the weight per attention mask could be lower than one, one is added to the weight mask such that the weight is always higher than one at each pixel. See figure 4.9 for an example

of a weight per attention map and the weight mask. The attention maps corresponding to the Kitti image are combined which results in the weight matrix. For the regions where attention maps give attention to the same region, an average weight is taken over these pixels. The calculated weight mask is multiplied by the per-pixel reprojection loss. Smaller objects will get more weight than larger objects since they contain a higher value in the weight mask. The hypothesis is that the depth prediction for objects further away will improve.

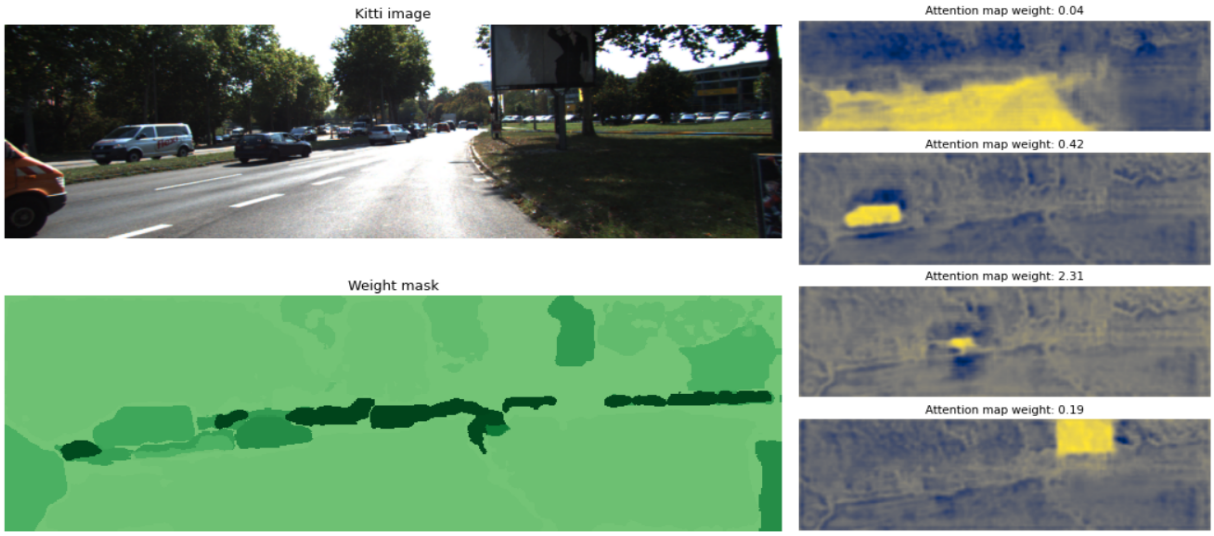

Figure 4.9: Kitti image with the corresponding weight mask. A subset of the corresponding attention masks with their weights is shown.

4.4 Transformer block

In this section, we explain the third experiment. The goal of this experiment is to improve the overall depth estimation by the use of a transformer block in combination with dilated convolutions. This is inspired by the work of (15).

Monodepth exists of two networks, one for converting the target image into the depth image and one for predicting pose parameters between the target and source frames (as explained in the background). In this experiment, only the depth network is updated. The monodepth depth network consists of an encoder-decoder architecture. In monodepth, first, the target image is sent to the depth encoder. This encoder encodes the image into five different features maps with different amounts of channels, width, and height. It uses 2D convolutions with batch normalization and a RELU as nonlinearity to create these feature maps. Then the feature maps are sent to the depth decoder which creates four different depth images at a different scale. The depth decoder does this by using upconvolutions (explanation follows). The depth images are upsampled afterward such that they have the same width and height as a Kitti image. In this experiment,

the last feature map of the depth encoder is updated with a combination of dilated convolutions and self-attention maps. See figure 4.10 for a schematic overview.

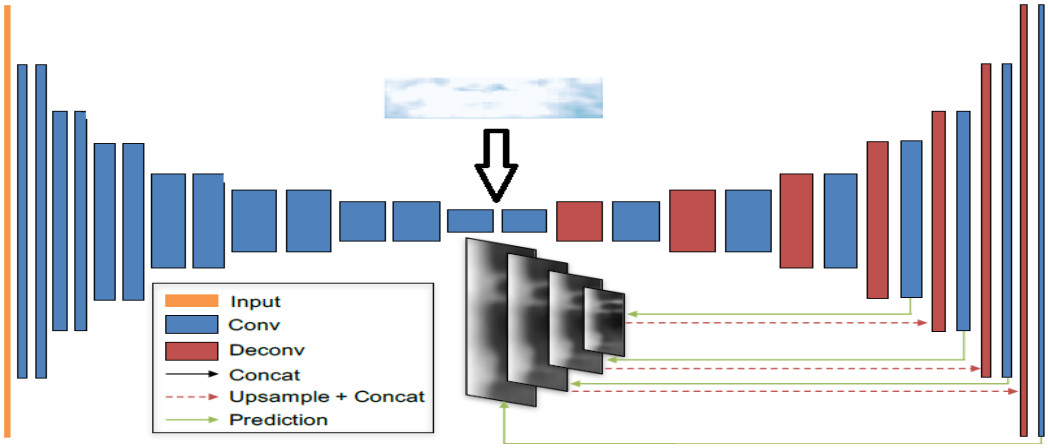

Figure 4.10: Overview of the updated architecture where the self attention maps with dilated convolutions are added at the latest stage of the encoder.

The self-attention maps are created with a combination of a transformer block and dilated convolutions. Three different dilated convolutions with dilation and padding size 12, 24, and 36 are created first. See section 3.3 for an explanation about dilated convolutions. The features from the dilated convolutions are mixed with the attention maps from the self-attention block (explained in section 2.3.2) by using a regular 2D convolution. The self-attention maps are added at the last encoder layer for two reasons. The first reason is that the last encoded layer has the smallest width and height, this makes it possible to add additional calculations and keep roughly the same training time. The second argument is that the last encoded layer will be used in all the decoding layers. In other words, all the decoding layers will make use of the added attention maps. Since the transformer block and the dilations have a larger receptive field than convolutions the hypothesis is that the decoder can benefit from this additional information. The attention maps give attention to either foreground, mid-ground, or background. This comes from the fact that the depth model gives similar depth values to the foreground, mid-ground, or background. See figure 4.11 for some examples. From a visual inspection, it appeared that most of the attention maps give attention to the background (sky). This is probably because the sky haves a uniform color along with the image and it is thus the easiest for the model to give attention to this region. This experiment hypothesizes that the depth decoder will benefit from these additional layers. Since these layers have a large receptive field they might help to encode the image into good features.

Figure 4.11: Kitti image with samples from the self generated attention maps. Blue indicate regions of attention.

Chapter 5

Experiments and Results

In this section, we will explain three experiments analyze their results. First, we explain the training settings and evaluation metrics. Next to that, per experiment, the preposessing steps are shown. At last, the result per experiment are analyzed.

5.1 Training settings

During training horizontal flips and the following augmentations are applied with 50 % probability: random brightness, contrast, saturation, and hue jitter with the following ranges: ± 0.2, ± 0.2, ± 0.2, ± 0.1. Important, all three images fed to the networks are augmented with the same parameters. Due to these augmentations, the results from monodepth are slightly different each run. The experiments will therefore be done three times to average out this randomization. The edge loss experiment was run with batch size 4 and the weight mask experiment and baseline model with transformer block was run with batch size 6. The same optimization steps from monodepth were followed (13). The experiments have been run on an 8B GPU.
Training monodepth takes 20 epochs following the original settings (13). This takes more than 60 hours for a single run on the GPU. To reduce training time but keep most of the results, early stopping is applied. Early stopping is determined based on three runs to see where the loss curve converged. Looking at figure 5.1 it can be seen that there is a decrease in the loss curves during the first 6 epochs. After this point, the loss curve stabilizes a bit. Therefore 6 epochs are determined as the early stopping point during training to reduce training time but keep most of the results.

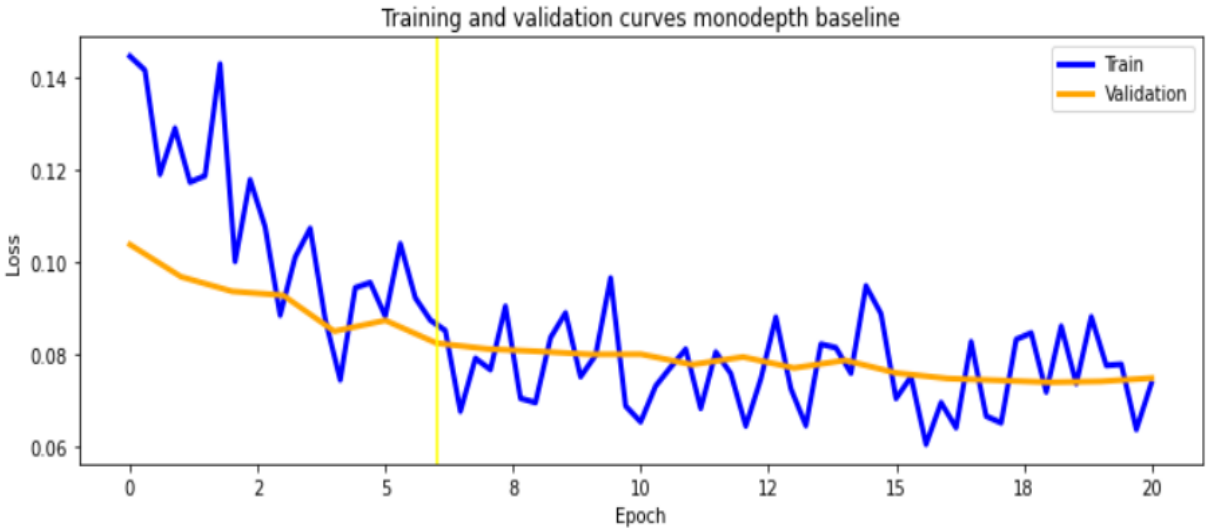

Figure 5.1: Training and validation curves to determine early stopping. The results are an average over 3 runs.

5.2 Evaluation metrics

Following the evaluation metrics by other self supervised depth estimation papers ((13), (11), (3), (33)), four different metrics are calculated when comparing the depth prediction with the depth labels. For all metrics, the mean value taken over the pixels is reported. Note that the metrics are only calculated over the areas where the depth label contains a depth value since the depth labels contain missing values.
GT means the ground-truth value which are the label values. $Pred$ are the predicted values from the model.

- Absolute relevance is the absolute per pixel difference divided by the label value.
  abs rel $= \frac{1}{N}\sum_{i=1}^{N}\frac{|GT_i - pred_i|}{GT_i}$
- Squared relevance is the per-pixel difference squared divided by the label value.
  sq rel $= \frac{1}{N}\sum_{i=1}^{N}\frac{(GT_i - pred_i)^2}{GT_i}$
- Root mean squared error is the root taken over the per-pixel difference squared.
  RMSE $= \frac{1}{N}\sum_{i=1}^{N}\sqrt{(GT_i - pred_i)^2}$
- Log root mean squared error is log taken over the root mean squared error.
  LOG RMSE $= \frac{1}{N}\sum_{i=1}^{N}\sqrt{\log(GT_i) - \log(pred_i))^2}$

These evaluation metrics will be used in this research to evaluate the results of the experiments.

5.3 Pre processing

To improve training time, some offline steps are performed before running the experiments. These steps are different for each experiment and will be described below per experiment.

5.3.1 Edge loss

Canny edge detection can be performed with different thresholds for finding edges with their magnitude. Three different thresholds were used on a subset of the data as an offline experiment. See figure 5.2 for an example. After this experiment threshold, 0.1 was chosen. This threshold does not capture the noisy edges but captures the right edges within an object.

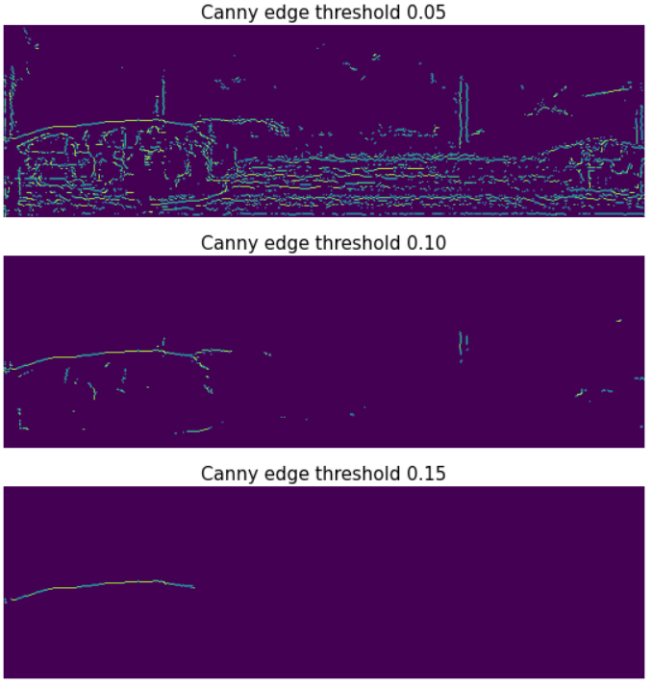

Figure 5.2: Different canny edge detection thresholds to detect edges on a depth image

For shrinking the attention map such that it only captures edges inside an object without the borders the erode function is applied. A kernel of size 3 by 3 with three iterations was used. This turned out to be the best fit of excluding object borders but remain the rest of the object as much as possible.
Before the edge loss is added to the original reprojection loss the edge loss is multiplied with a weight of 2e-2 to give the edge loss roughly the same value as the range of the original loss. This value was chosen after a few runs with different weight values. Because the depth network needs to learn at the start of the training it is not fair to introduce the

edge loss from the beginning. The edge loss will find a lot of edges which is because the depth network is still learning how to predict depth. Therefore the edge loss is introduced after the first epoch of training. It turns out that the edge loss behaves normally after one epoch. This is based on a test run where the edge loss was introduced from the beginning. See figure 5.3 for the loss curve during training. It is visible that the edge loss is taking over the total loss function in the first epoch. This is because the depth network first needs to learn depth. If the edge loss will be introduced from the beginning this will give a lot of noise to the model and it might not learn well to predict depth in general. after the first epoch, the edge loss behaves normally, therefor during training the edge loss will be introduced after the first epoch.

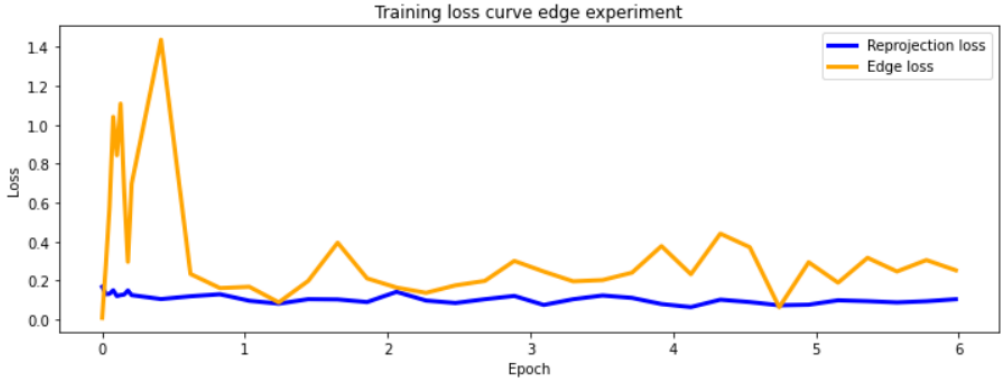

Figure 5.3: Test run of the loss curve during training when the edge loss was introduced from the beginning

5.3.2 Weight mask loss

The attention masks from the DETR model were collected before training. The advantage is that during training the GPU is not filled with the DETR model and the attention maps are already calculated. An important note is that the prepossessing steps do not influence inference. The weight mask is only used during training and after training only the trained depth network is needed to convert color images to depth images.

As mention before, each attention mask comes with a probability of how confident the DETR model is. The higher the probability the more confident the model is that it correctly gives attention to a single object. In this experiment, only attention masks with a probability higher than 0.5 were used. We found that this is the right balance between qualitative good attention masks and also a broad range of objects which got attention. After the collection of the attention masks per Kitti image, the weight mask matrix is calculated as described in section 4.3. Before the weight mask is multiplied times the reprojection loss during training the following pre-processing steps are applied. First the pixels in the weight mask with a value ≤ 1.05 are mapped to 1. High values in the weight mask matrix indicate that they will cover an object which contains a small

object. The value 1.05 is chosen such that bigger objects are mapped to 1 such that their weight does not influence the reprojection loss. The values > 1.05 are multiplied by 5. Pixels > 1.05 are part of smaller objects and they are multiplied times 5 such that they get a higher value. Both 1.05 and 5 are chosen with a small offline experiment to visually see differences. Due to limited GPU power and time it has not been possible to experiment with different values. At last, the reprojection loss is multiplied by 0.8 such that the reprojection loss shrinks. Now the updated weight mask is multiplied times the updated reprojection loss. See figure 5.4 for these steps. Looking at the first row it is visible that the relative larger objects are mapped back to 1 and the smaller objects get more weight. It is visible that the reprojection loss multiplied with the weight mask is different from the original reprojection loss (bottom left vs top right). The objects further away in the scene (smaller objects) get more loss and the reprojection loss outside these regions is decreased a bit.

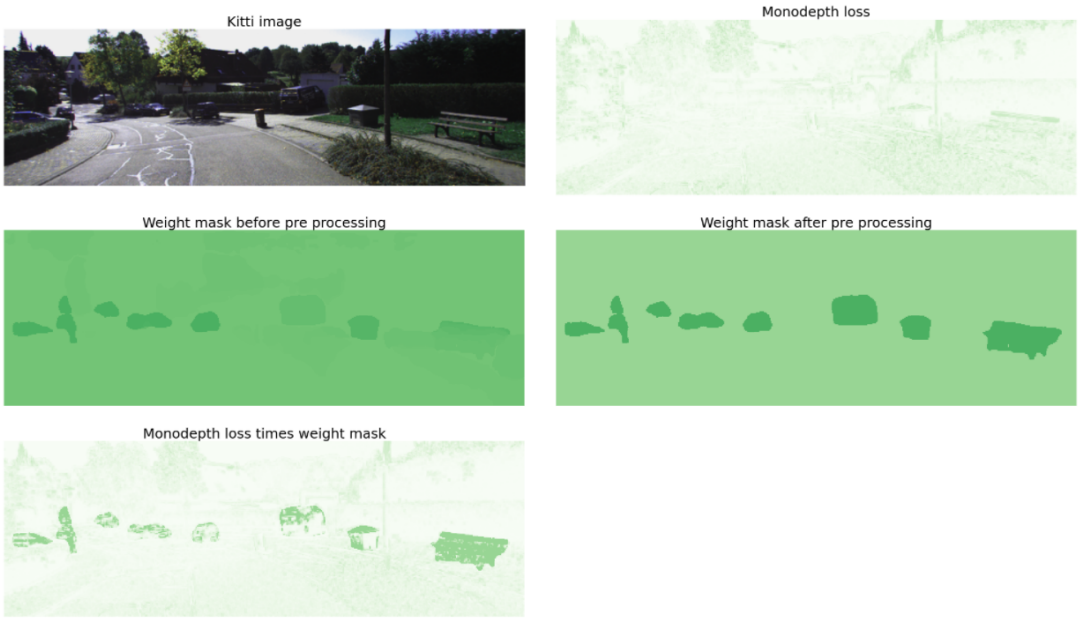

Figure 5.4: Example of the pre-processing steps of the weight matrix and result after multiplication during training. Monodepth comes with a per-pixel loss (right up). This is multiplied by 0.8. The weight mask is adjusted such that smaller objects get more weight and larger objects are ignored (middle row). Reprojection loss is multiplied by the adjusted weight mask (bottom left) and is used for backpropagation.

5.3.3 Transformer block

Since this experiment only updates the architecture of the depth encoder no pre-processing steps had to be made with the dataset.

5.4 Edge loss

The results section of this experiment contains two parts. First, the training's loss and visual differences between depth images are analyzed. Second resulting evaluation metrics are discussed based on the depth labels.

5.4.1 Reprojection loss

First, it is interesting to see how the loss of the training behaved during training. Figure 5.5 shows the training's loss. This loss curve includes the reprojection loss and the additional edge loss. It is visible that the edge loss is started after the first epoch as explained in section 5.3.1. The regular reprojection loss is normally around 0.15 which means that the edge loss roughly doubles the loss during training. This indicated that the loss functions find quite a lot of edges during training.

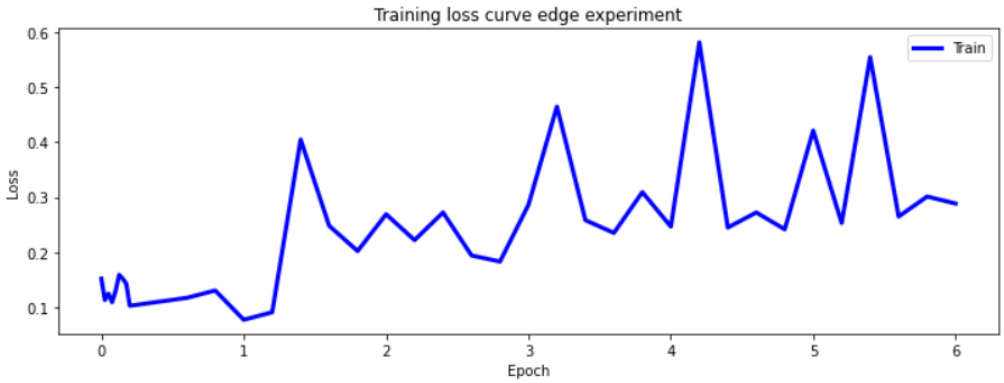

Figure 5.5: Training loss during training (reprojection and edge loss)

To investigate if the depth images look different analysis is performed on the depth images from the test set (n = 700). The structural similarity index (SSIM) and the absolute pixel differences are calculated between the depth images from the baseline monodepth model and the trained model with the edge loss. For an explanation about how SSIM works, view section 2.2.1. The higher the SSIM score, the more similar the images are. When looking at table 5.1 first the three baseline models are compared since there is a bit of randomness in the monodepth models. It turns out that there is almost no difference in the SSIM scores between different baseline runs. As second the depth images from the edge experiment are compared with the baseline models. It turns out that the SSIM between these images on average is 0.88 whereas the difference between the baseline models is 0.90. This indicated that there are small differences between the depth images. The absolute pixel-wise differences between the depth images from table 5.2 give a similar result. Between the baseline runs there is almost no difference, the edge experiment gives a small difference indicating that the depth images look a little bit different. Note that the absolute differences can range from 0 to 6 since the depth images are bounded between these values due to visualization reasons.

SSIM	Baseline 1	Baseline 2	Baseline 3	Edge experiment
Baseline 1	x	0.89 +/- 0.04	0.90 +/- 0.04	0.88 +/- 0.04
Baseline 2		**x**	0.90 +/- 0.04	0.88 +/- 0.04
Baseline 3			x	0.88 +/- 0.04

Table 5.1: SSIM between the depth images on the test set (n = 700) between the baseline model and the edge experiment model. The numbers after the +/- sign indicate standard deviation.

ABS Diff	Baseline 1	Baseline 2	Baseline 3	Edge experiment
Baseline 1	x	0.22 +/- 0.13	0.21 +/- 0.12	0.31 +/- 0.14
Baseline 2		**x**	0.20 +/- 0.10	0.36 +/- 0.17
Baseline 3			x	0.30 +/- 0.14

Table 5.2: Absolute difference between the depth images on the test set (n = 700) between the baseline model and the edge experiment model. The numbers after the +/- sign indicate standard deviation.

It is interesting to compare the biggest difference in the depth images between the edge experiment and the baseline models. In figure 5.6 three examples can be found where the difference between the models is the maximum. The first example indicates that the car windows for some cars now have roughly the same depth as the car. This is correct because the car window is roughly equally far away from the other parts of a car. The differences again are small. But it must also be said that not all car windows received the same depth as the car as can be seen in the background cars of the first example. This indicated that the loss term does not help enough to solve the depth estimation of reflective/transparent objects. The second image shows that for some objects the depth is similar to the depth on the background which is a bad sign. Due to the edge loss, other edges are found and therefore the model sometimes predicts objects the same as their background. The third example shows that the model is better in predicting sparse objects indicated by the pole. Note that these example images contain the maximum differences. Most depth imaged does not contain any difference.
To investigate if the edge loss helps the model in reconstructing the target image the reprojection error is analyzed. The reprojection error results are calculated over the test set (n = 700). The reprojection error indicates how good the model is in reconstructing the target image via the source images by using the depth and pose predictions. See table 5.3 for the reprojection loss. It can be seen that the reprojection error is almost the same.

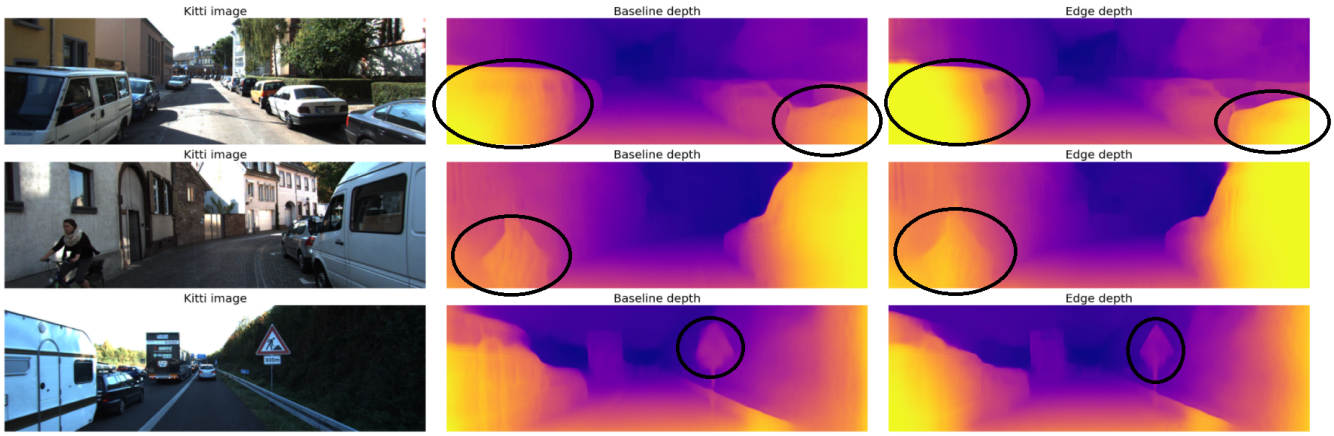

Figure 5.6: Depth image examples from the baseline models (middle column) with the highest differences compared to the edge experiment (right column). Differences indicated by the black circles.

Model	Reprojection loss
Baseline	0.093
Edge experiment	0.090

Table 5.3: Average reprojection error after 6 epochs of training calculated on the testset (n = 700). Lower is better.

5.4.2 Depth labels

When the depth images from the models are compared to the depth images coming from the sensors it gives a similar result (see table 5.4) as the reprojection loss. The edge loss does not seem to influence the depth images when looking at the evaluation metrics (Abs Rel, Sq Rel, RMSE, and RMSE log) results in the table. Together with the SSIM and absolute differences results on the depth images in table 5.1 and 5.2 it can be concluded that the edge loss is almost not influencing the baseline model.

Model	Abs Rel	Sq Rel	RMSE	RMSE log
Baseline	0,120	0,919	4,846	**0,194**
Edge experiment	0,120	**0,903**	**4,838**	0,196

Table 5.4: Model results based on the depth images coming from the models compared to the depth images coming from the sensors. Results are based on the testset (n = 700). Lower is better.

5.5 Weight mask loss

The results section of this experiment consists of two parts. First, the reprojection loss (how good are the reconstructed images) during training is analyzed for different areas. Second, the depth labels will be used to calculate how good the depth images are. Based on the depth labels the evaluation metrics from section 5.2 can be calculated.

5.5.1 Reprojection loss

First, we look at the reprojection loss on the test dataset during training. The test set is never seen during training or touched before. The lower the loss the better the model is in reconstructing the target image. The models have been trained for 6 epochs (early stopping) on the training set and three runs per experiment have been performed to average the randomness. The loss is calculated over the test dataset (n = 700).
Looking at figure 5.7 it can be seen that the loss for the whole image (left up) is similar to the baseline where the weight loss was added. This means that the monodepth model still performs well when the additional loss function was added. Given the fact that the reconstruction for the whole image remains the same when the weight mask loss is added compared to the baseline, it is interesting to see if the regions inside the weight mask benefit from the additional weight mask loss. The loss inside small objects is calculated by filtering the weight mask where only pixels > 1.05 to 1 are used to calculate the loss inside the mask. Because pixel values > 1.05 cover small objects as explained in section 5.3. This updated weight mask is multiplied times the per pixel reprojection loss and a mean is taken over this area. Looking at the loss inside the weight mask for small objects (right up in figure 5.7) it can be seen that the weight mask loss is having a lower loss than the baseline. This means that the model is better in reconstructing the target image for the regions where the loss is applied and still performs well for the other parts. It can be concluded that the additional loss function helps. Figure 5.8 shows some depth images where it is visible that the model can predict depth for objects further away, whereas the baseline model is not capturing these objects. Again indicating that the model is benefiting from the additional loss function. To double-check if the model performs well outside of the regions of the small object, dilation is added to the weight mask with a kernel of ones with the size 2 by 2. The original weight mask is then subtracted from the weight mask with dilation such that only pixels outside of the weight mask remain. See figure 5.9 for a visual example. Looking at the bottom line graphs in figure 5.7 including dilation it can be seen that the model behaves normally just outside the small object regions when 1 and 3 rounds of dilations are added to the weight mask. It can thus be concluded that the models still behave well just outside the weight mask.

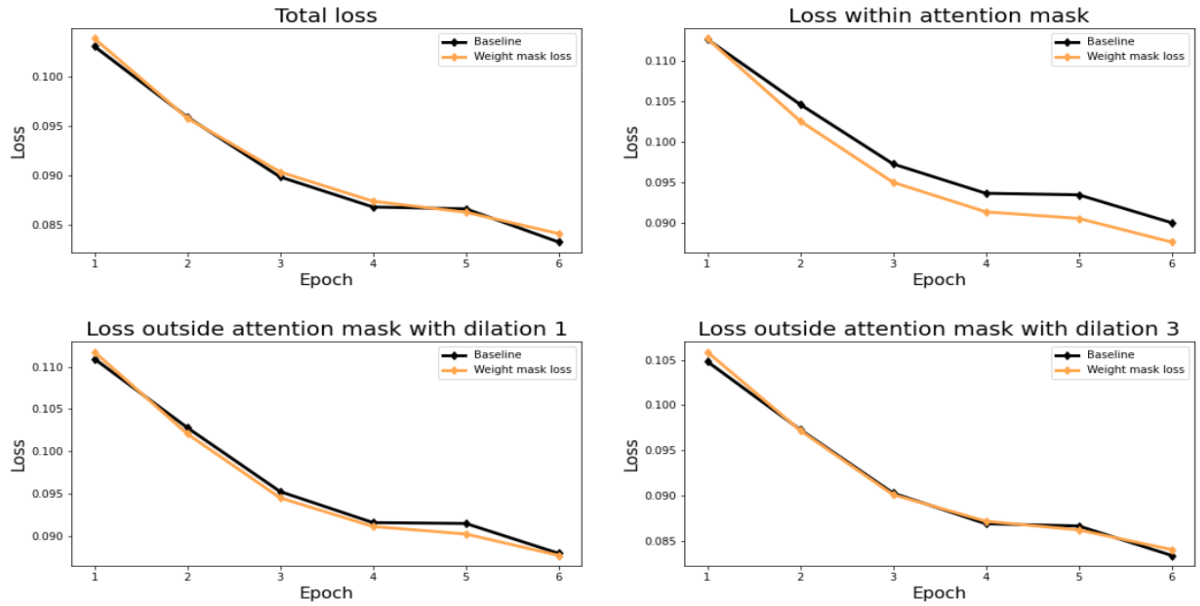

Figure 5.7: Reprojection loss over the test dataset (N = 700) for different epochs. The values are an average over 3 runs.

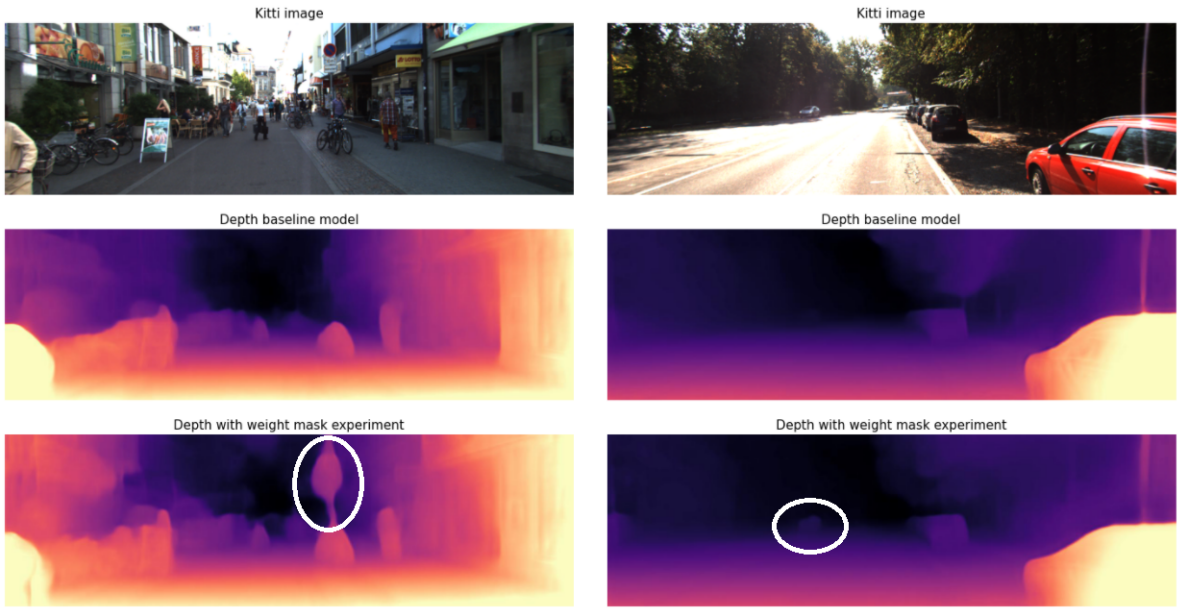

Figure 5.8: Depth images where the weight mask loss helps the model predicting depth for objects far away from the camera.

A second analysis is performed on the reprojection loss. In this analysis images containing different amounts of small objects are investigated to see how much the model

Figure 5.9: Top: the Kitti image with the per-pixel reprojection loss at the beginning of training. The left column shows the weight mask after pre-processing, and with 1 and 3 rounds of dilation added. The right column shows the loss inside these masks on the left.

benefits from the additional loss function. To calculate how many small objects an image contains, the corresponding weight mask is filtered. The number of values (pixels) in the corresponding weight mask > 1.05 are counted. Since small objects get a high value in the weight mask based on the DETR attention masks as explained in section 4.3. See figure 5.10 for a Kitti images which is covered with $> 10\%$ of small objects and $< 1\%$. The analysis can be seen in figure 5.11. It can be seen that the reprojection loss with the weight mask loss benefits the most when images contain relatively a lot of small objects, +/- 10 %. Images with a less amount of smaller objects in the image result in roughly the same reprojection loss. It thus means that the model benefits the most when an image is at least covered by a small object with 10 %. Unfortunately, the Kitti dataset does not contain a lot of images that are covered by >10% with small objects indicated by the bottom graph in figure 5.11. This is the reason why the total reprojection loss doesn't benefit from this additional loss but only the reprojection loss inside the small objects as indicated by the line graphs in figure 5.7

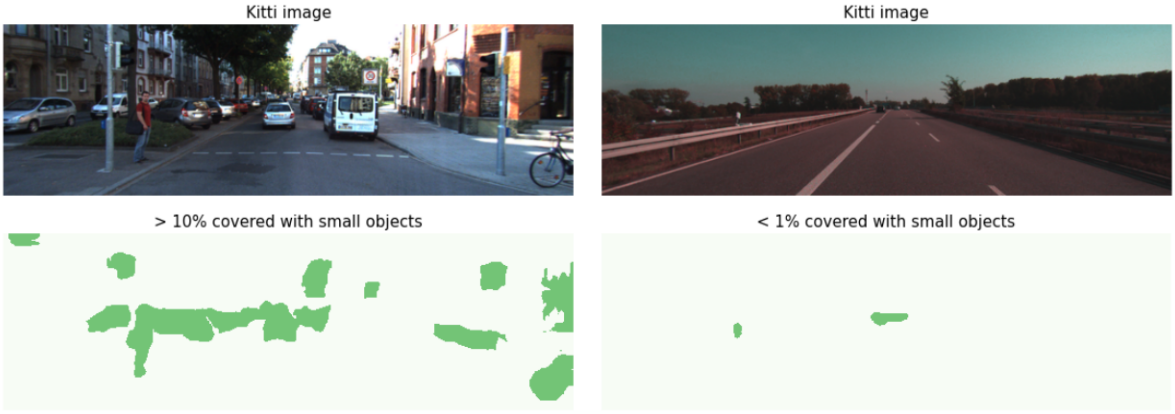

Figure 5.10: Example of a Kitti image which is covered with $>$ 10% of small objects (bottom left) and $>$ 1% (bottom right).

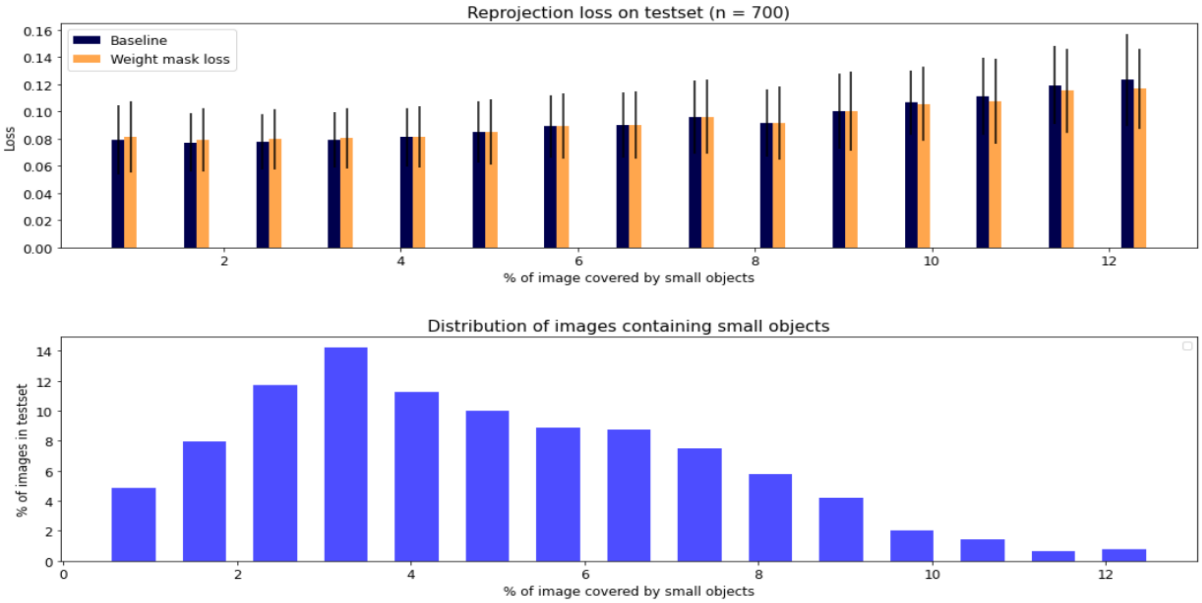

Figure 5.11: Reprojection loss on the test dataset (n = 700) for images with different amounts of small objects. Top: The mean reprojection loss for the whole image. The vertical bars indicate the standard deviation. Bottom: The number of images containing small objects. The values are an average over 3 runs. Note to read the bottom plot. For example how to read the highest bar in the image. 14% Of the images in the test set are covered with roughly 3% with small objects.

5.5.2 Depth labels

The test set of the Kitti dataset contains depth labels (700 images). The drawback of these labels is that they are made with a lidar sensor. Therefore the labels are noisy and they contain a lot of zero values. See figure 4.3 for some examples. The results from this section are thus less confident than the reprojection loss section discussed above. The advantage of these labels is that a lot of the self-supervised papers publish their results based on these labels (for example: (13), (11), (3), (33)), so a comparison can be made. In this paper, the comparison is made with the monodepth (13) model (baseline).
Looking at the evaluation metrics (Abs Rel, Sq Rel, RMSE, and RMSE log) results in table 5.5 it can be seen that the weight mask loss performs slightly worse than the baseline. This is a different result as the reprojection loss from figure 5.7 where both models performed similarly. These differences might partly be because the label values are noisy and contain missing values. Another reason why the overall performance is slightly worse is that the model is paying less attention to the objects close by. Sometimes object close by are not captured by the model when the weight mask is applied. Since objects close by contain relatively more pixels than objects further away, this has a bigger influence on the depth labels. See figure 5.12 below for an example.

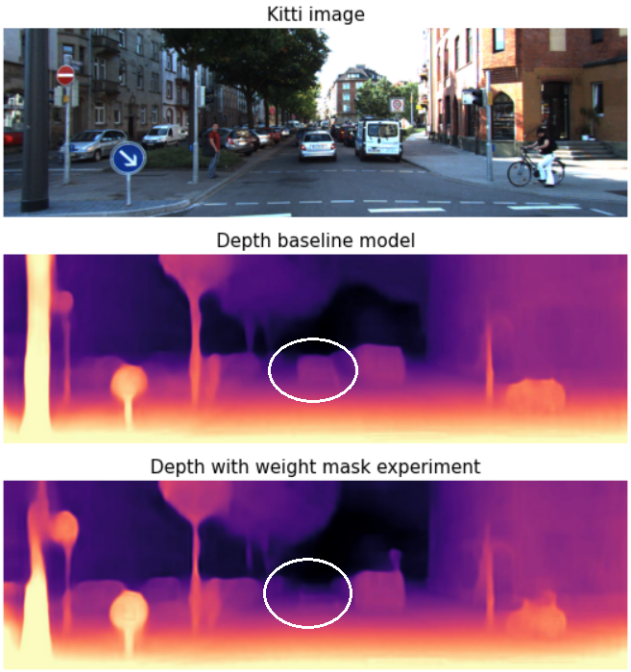

Figure 5.12: Example where an object close by is not captured in the depth image due to the weight mask loss.

The second table (table 5.6) analyses the label results for the areas in the images covering small objects. This is calculated by filtering the corresponding weight mask where only pixels values > 1.05 are used to calculate the loss inside the mask. Because pixel values > 1.05 cover small objects as explained in section 5.3. See figure 5.13 for a visual example. When looking at the results of the label values inside small objects in table 5.6 it can be seen that the weight mask loss is performing slightly worse than the baseline for the areas containing small objects. This is unexpected since these areas got more loss during training due to the extra loss function. For the reprojection loss, it can be seen that the weight mask loss was performing better for the regions covering small objects. This indicates again that the label values are noisy and contain missing values. It is thus difficult to draw conclusions based on these label images.

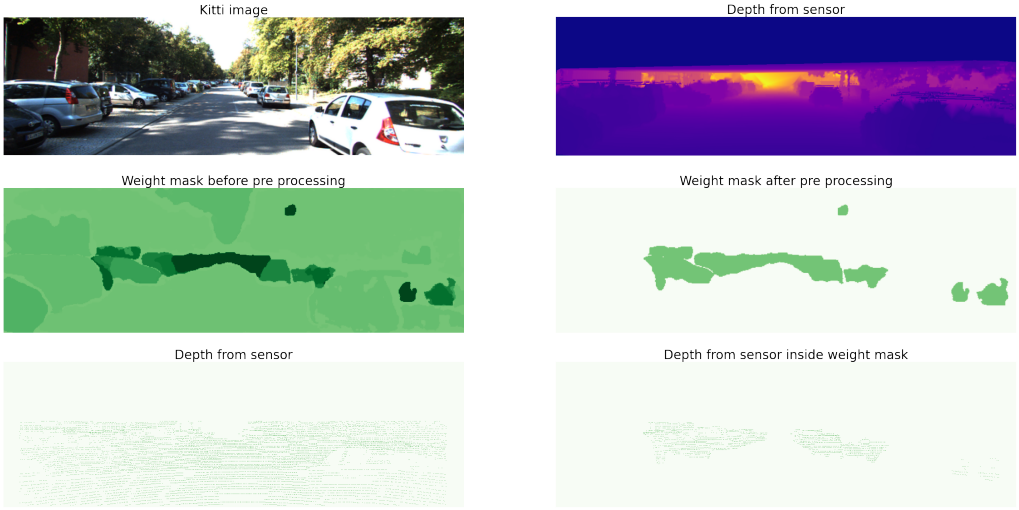

Figure 5.13: The weight mask after pre-processing is multiplied by the depth from the sensor to select the depth values covering small objects. The depth from the sensor is shown twice with different color maps to show that part of the depth values are missing.

Model (Whole image)	Abs Rel	Sq Rel	RMSE	RMSE log
Baseline	**0.120**	**0.919**	**4.846**	**0.194**
Weight mask loss	0.125	0.980	4.947	0.197

Table 5.5: Model results after 6 epochs (early stopping). The depth prediction is compared with the label where the whole image is taken into account. The numbers are averaged over 3 runs due to model randomization.

Model (Inside weight mask)	Abs Rel	Sq Rel	RMSE	RMSE log
Baseline	**0.198**	**2.826**	**8.519**	**0.273**
Weight mask loss	0.201	2.965	8.701	0.279

Table 5.6: Model results after 6 epochs (early stopping). The depth prediction is compared with the label where only regions inside the weight mask are taken into account. The numbers are averaged over 3 runs due to model randomization.

5.6 Transformer block

The monodepth model is being trained with the updated depth network. The depth encoder now includes dilated convolutions and self attention maps (transformer block) as described in method section 4.4
Two different experiments are performed. The first experiment is the Baseline model with the transformer block. The second experiment is the Baseline model with the transformer block in combination with the weight loss as described in section 2.3.2. Where the idea is that smaller objects get a higher reprojection loss based on the DETR attention masks. The goal is that these regions get a better depth prediction. Note that the added weight loss function is still based on the DETR attention maps and not based on the self-generated attention maps. This is because the DETR attention maps capture different objects since DETR was trained on object classification. The attention maps generated in this research are trained on a self-supervised depth estimation network and give attention to either foreground, midground, or background as can be seen in figure 4.11. The attention maps are thus fundamentally different and therefore it is not possible to use the self-generated attention maps to calculate a weight mask as was done in section 2.3.2 with the DETR attention maps. The expectation is that the baseline model with transformer block will improve the overall model performance and the transformer block with weight mask loss will perform better for regions where the weight mask is applied (small objects).

5.6.1 Reprojection loss

In figure 5.14 the reprojection loss can be found for the two experiments' transformer block and transformer block with weight mask loss. The baseline model with transformer

block results in a slightly higher reprojection loss compared to the baseline. This indicates that the model is not benefiting from the transformer block. It might be that the transformation block and the dilated convolutions are introduced at a too deep stage in the encoder. Since these blocks are introduced in the last part of the encoder it might be that the encoder already lost important details which is a common problem in convolutional encoders. Introducing the transformer block and dilated convolution also in the other layers might improve the results. When the weight loss is added to the transformer block the loss remains the same indicating that again the model remains the same. When looking at the loss within the attention mask it is again visible that the weight mask loss in having a lower loss for the transformer block with weight mask loss than the only the transformer block. This indicated that the model benefits from this additional loss function. The loss outside the regions of the weight mask performs similarly as inside the weight mask indicating that the model performs still well around these regions.

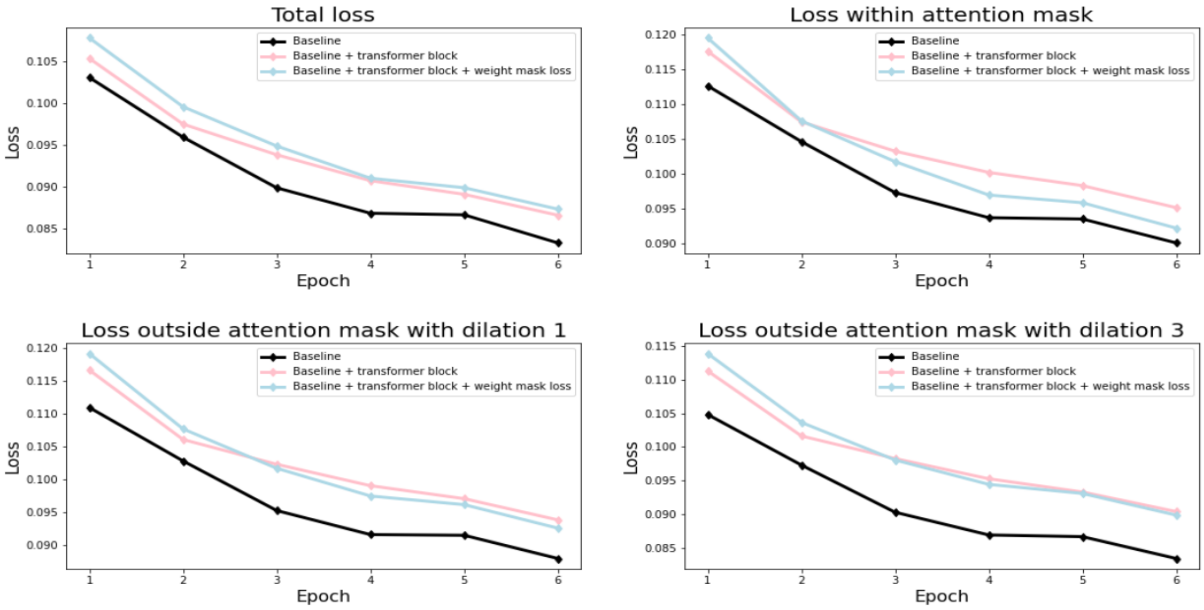

Figure 5.14: Reprojection loss over the test dataset (N = 700) for different epochs. The values are an average over 3 runs.

A interesting difference is found between the self-attention maps the model generated during training between the two experiments. It turns out that when the baseline model with transformer block is trained with the weight mask loss the attention masks are more focused on smaller objects. The attention maps give only attention to the smaller object and not to other regions. The attention maps without this loss give attention to several regions in the image. This indicates that the loss function is influencing the self-attention maps and therefore the model can benefit from these maps. See figure 5.15 for

some differences between the attention maps generated with and without the additional loss function.

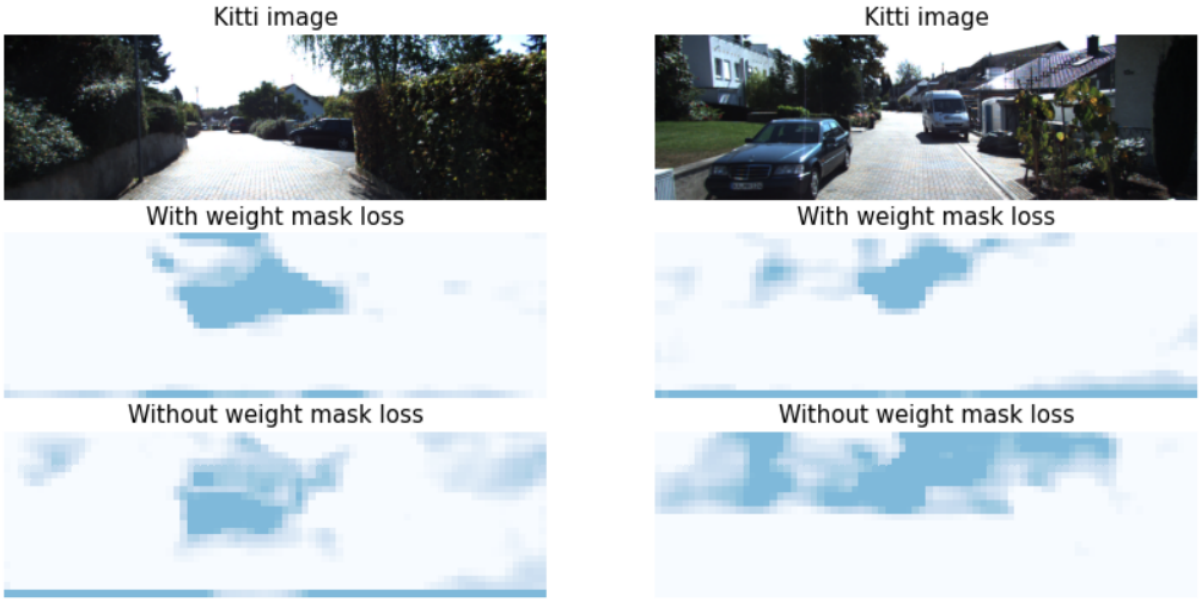

Figure 5.15: Self generated attention maps with and without the weight mask loss

5.6.2 Depth labels

First, we look at the evaluation metrics (Abs Rel, Sq Rel, RMSE, and RMSE log) results in table 5.7 where the whole depth image is taken into account. These results are based on the depth images from the model compared to the depth images coming from lidar from the test dataset. It can be seen that the model can benefit from the transformer block since it slightly outperforms the baseline. This thus indicates that the model benefits from the dilated convolutions in combination with the transformer block. The experiment with the transformer block and the weight mask loss results in a similar result. This is probably because the Kitti dataset does not contain enough images with small objects to create a difference. Again it must be said that the lidar images are noisy and contain missing values so it is difficult to draw conclusions based on them.
As second we look at how the models perform for the regions containing smaller objects, these results can be found in table 5.8. It can be seen that the baseline model with transformer block without the weight loss is the best model when following the lidar depth images. This is a surprising result because the reprojection loss from figure 5.14 shows that the model with the transformer block with the weight mask loss is performing

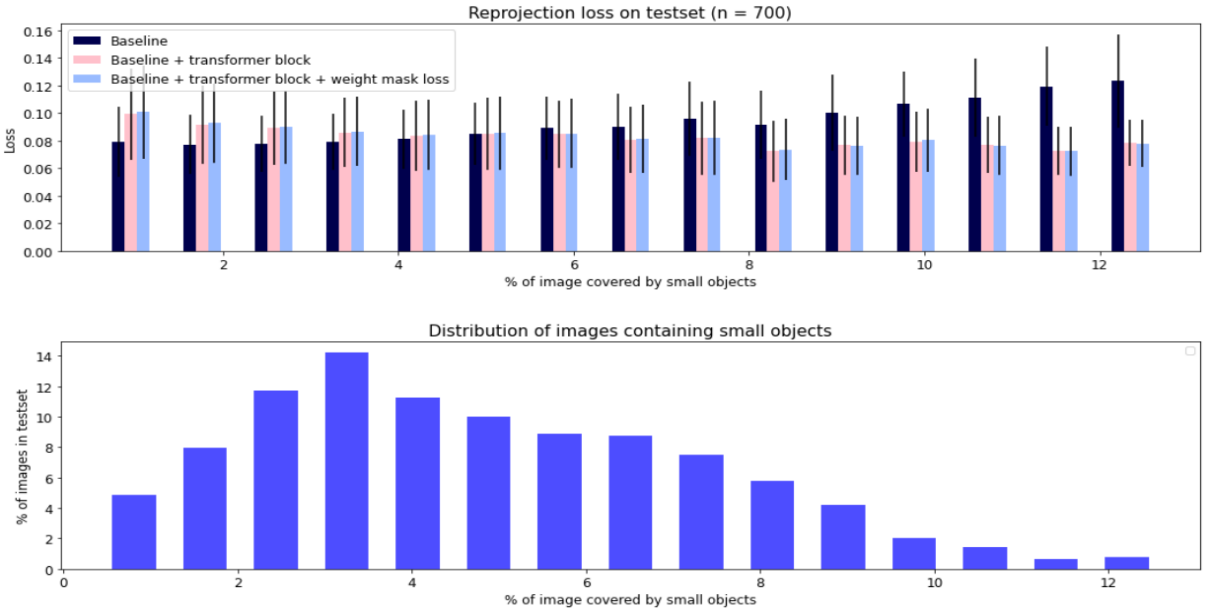

Figure 5.16: Reprojection loss on the test dataset (n = 700) for images with different amounts of small objects. Top: The mean reprojection loss for the whole image. The vertical bars indicate the standard deviation. Bottom: The number of images containing small objects. The values are an average over 3 runs. Note to read the bottom plot. For example how to read the highest bar in the image. 14% Of the images in the test set are covered with roughly 3% with small objects.

Model (Whole image)	Abs Rel	Sq Rel	RMSE	RMSE log
Baseline	0.120	0.919	4.846	0.194
Baseline + transformer block	**0.119**	**0.890**	**4.811**	**0.193**
Baseline + transformer block + weight mask loss	**0.119**	0.954	4.934	0.196

Table 5.7: Model results after 6 epochs (early stopping). The depth prediction is compared with the label where the whole image is taken into account. The numbers are averaged over 3 runs due to model randomization.

the best. The reason why the result from the lidar information is different is probably that these images contain a lot of missing values, therefore we prefer to look at the reprojection loss. Overall it can thus be said that the transformer block works best for images containing a lot of small objects. But since the Kitti dataset does not contain a lot of these images the transformer block is not usable for this dataset.

Model (Inside weight mask)	Abs Rel	Sq Rel	RMSE	RMSE log
Baseline	0.198	2.826	8.519	0.273
Baseline + transformer block	**0.190**	**2.640**	**8.381**	**0.269**
Baseline + transformer block + weight mask loss	0.201	3.124	8.754	0.282

Table 5.8: Model results after 6 epochs (early stopping). The depth prediction is compared with the label where only regions inside the weight mask are taken into account. The numbers are averaged over 3 runs due to model randomization.

Chapter 6

Discussion

In this section, we discuss the experiment and results. Starting with the discussion of the first experiment, where the goal was to reduce the depth gaps of reflective/transparent objects. Second, the weight mask experiment will be discussed where the goal was to improve the depth estimation of small objects. Third, the experiment with the transformer block and dilated convolutions will be discussed where the goal was to improve the overall depth estimation. At last two problems of the self-supervised depth estimation field are discussed.

First we look at the edge loss experiment where the goal was to reduce the depth gaps of reflective/transparent objects (car windows for example). The problem with reflective and transparent objects is that this is inherently a problem when performing an image reconstruction task (what self-supervised depth estimation is). There is an ongoing discussion that a transparent/reflective surface has two different depths, namely the depth of the surface itself (1) and the depth of the scene behind the surface (2) (objects behind the transparent object). In depth estimation, the depth of the surface itself is the correct depth since you can't go through the surface. But the current self-supervised depth estimation models estimate the second depth type, the objects behind the surface since these objects are transparent/reflective. The edge loss gives a loss to single objects which have a big difference in depth based on the DETR attention maps. The goal was that reflective and transparent objects got the depth of the object itself instead of the scene behind the object. Unfortunately, some DETR attention maps covered multiple objects such that some object boundaries were captured as edge as well. The edge loss was highly influenced by these mistakes. As a result, there was almost no visual difference between the depth images from the baseline model and the depth images from this experiment based on absolute differences and SSIM scores. It would help if the attention masks from DETR would have a corresponding object class. Then we could perform the same experiment applied on only a specific subset of the attention maps of objects which contain these kinds of surfaces. (15) Also mention in their self-supervised depth paper that it is difficult to solve the depth for reflective/transparent objects and mention that this is one of the problems of self-supervised depth estimation via image reconstruction.

As second, we look at the experiment with the weight mask loss where we tried to improve the depth estimation for objects further away from the camera. The results for reconstructing the target image are better for the regions which are further away from the camera. This indicates that the model benefits from the loss function. Unfortunately, the reprojection loss for the whole image was not improved but remained the same. This might be due to two reasons. First, the Kitti dataset does not contain enough images with small objects in the background such that the whole model can benefit from this additional loss. Looking at the top plot of figure 5.11 it can be seen that only images which are covered with more than 10% of small objects can benefit from the loss function. These images have a lower reprojection loss for the whole image. Based on the test set, only roughly 8% of the test set contains these kinds of images as can be seen in the bottom plot of figure 5.11. This is unfortunately not enough such that the model can benefit from this. The second reason why the overall reconstruction loss probably did not improve is because with this additional weight loss the model is focusing more on the background. Therefore, sometimes the model is performing worse on objects in the for ground as can be seen in figure 5.12. Since objects in the foreground contain relatively more pixels, this has more influence on the loss than objects in the background. An idea to improve the overall reprojection loss might be to train the model on a dataset that contains more smaller objects. (23) Generated their own dataset by training depth estimation on roughly 20 movies containing over 75.000 frames. Probably movies that take place inside will contain more small objects compared to the Kitti dataset which is taking place outside. The model could then benefit from the additional loss function on indoor movies. A second idea that might work for this experiment is to only backpropagate this additional loss through the depth network and not through the pose network. Since this loss is based on better depth estimation for smaller objects further away this loss function might be noise for the pose network. In this research, we choose to backpropagate the loss through both networks to investigate if both networks might benefit from this additional loss.

At third we look at the last experiment where a transformer block with dilated convolutions was added to the baseline architecture. The goal was to improve the overall depth estimation with this updated architecture. The transformer block has a global receptive field and the dilated convolutions have a large (size 12 to 36) receptive field. Since convolutions have a limited receptive field, and depth estimation is a dense prediction, the model might benefit from these additional resources. Looking at the results from the depth labels (table 5.7), the baseline model with transformer block performs slightly better than the monodepth baseline. But since the depth label values are sparse and noisy it is difficult to draw conclusions on these results. Looking at the results from the reprojection loss on the test dataset (figure 5.14) it turns out that the transformer block performs slightly worse. A reason for this result might be that the transformation block and the dilated convolutions are introduced at a too deep stage in the encoder. Since these blocks are introduced in the last layer of the encoder it might be that the encoder

already lost important details which is a common problem in convolutional encoders. Introducing the transformer block and dilated convolution also in the earlier layers, might improve the results.

In this research it has not been possible to perform a grid search per experiment over the hyperparameters due to a long-running time per experiment. The hyperparameters in the experiments were chosen based on short offline experiment runs. As the experiments and models contain a lot of hyperparameters, it might be that the chosen hyperparameters prevented the model from reaching optimal results.

At last, two problems of the self-supervised depth estimation field will be discussed based on this research:

- The self-supervised depth estimation papers should rely less on the lidar depth information coming from sensors that are used as ground truth. All the papers compare their depth estimations with these lidar depth images. This information contains a lot of missing values and a lot of values are noisy. From a research perspective, it is thus not fair to compare models which each other based on this insecure information. A possible solution could be that the papers publish theirs scores on how good the reconstruction images are after training. Since a low reconstruction loss indicates a good pose model and a good depth model. A score could be a combination of the absolute pixel-wise difference and SSIM between the target image and the reconstructed target image.
- As second, a lot of models in the self-supervised depth estimation field are performing slightly differently on different runs. None of the papers mention this and none of the papers publish their results based on average runs. All of the papers only publish their best score based on a single run. The published depth images are based on this best run. These results are thus biased and do not give a completely fair overview. There should be an agreement within the field that the published results should be average results instead of best run based on randomness.

Chapter 7

Conclusion

This thesis presented novel research that combined a self-supervised depth estimation model with a transformer model. Next, we provide some conclusions based on the performed experiments.

- The edge loss experiment tried to reduce the depth gaps of reflective/transparent objects. The additional loss function does not seem to have a notable difference when comparing depth images, the reprojection loss, and the depth labels between the baseline and this experiment.
- The weight mask loss tried to improve the depth estimation for small object. The loss function resulted in a lower reconstruction loss for the regions containing smaller objects. This was also visible in some depth images where objects further away from the camera were captured due to this loss function.
- The weight mask loss does result in a similar reconstruction loss for the whole image due to two reasons. First, the Kitti dataset does not contain enough images with small objects. When this loss function would be applied on for example indoor movie scenes with a lot of small objects, the overall model will probably benefit from this. Second, due to this additional loss function, the model is focusing more on the background whereby the depth estimation of the for ground decreased for some objects.
- The baseline model with transformer block and dilated convolutions tried to improve the overall depth estimation. These additional layers only seems to help for Kitti images which are covered with more than 6% with small objects. Since only roughly 20% of the Kitti dataset contains these kinds of images, these additional layers are not helping the overall score. Introducing the dilated convolutions and transformer block in earlier layers of the depth encoder as well might improve the results. A disadvantage of that approach is that is would increase the amount of model weights.

Further research

Transformer models seem to replace and outperform traditional convolutional based models in the computer vision field. (10). This is slowly also the case for the self-supervised depth estimation field (15). Therefore, as further research the transformer block could further replace the convolutional blocks with transformer blocks at earlier layers of the model. These architectures already outperform the convolutional models in tasks like object detection and classification (10), (21), (1). As second, the weight mask loss could be applied to an indoor movie dataset that contains more smaller objects compared to the Kitti dataset. Probably the whole model will then improve from this loss function. A third future work would be to apply the edge loss only on attention maps that cover transparent/reflective objects. Currently, transformer models are not able to classify attention masks.

Bibliography

[1] Bello, I; Zoph, B.: Attention augmented convolutional networks. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 3286-3295). (2019)

[2] Canny, J.: A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence, (6), 679-698 (1986)

[3] Carion, N.; Massa, F.S.G.: End-to-end object detection with transformers. In European Conference on Computer Vision (pp. 213-229). Springer, Cham. (2020)

[4] Casser, V.; Pirk, S.M.R.A.A.: Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos. In Proceedings of the AAAI conference on artificial intelligence (Vol. 33, No. 01, pp. 8001-8008). (2018)

[5] Chen, L; Papandreou, G.: Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE transactions on pattern analysis and machine intelligence, 40(4), 834-848 (2017)

[6] Chen, L; Papandreou, G.: Rethinking atrous convolution for semantic image segmentation. arXiv preprint arXiv:1706.05587. (2017)

[7] Chen, C; Zhu, Y.: Encoder-decoder with atrous separable convolution for semantic image segmentation. In Proceedings of the European conference on computer vision (ECCV) (pp. 801-818) (2018)

[8] Chen, L; Zhu, Y.: Encoder-decoder with atrous separable convolution for semantic image segmentation. In Proceedings of the European conference on computer vision (ECCV) (pp. 801-818). (2018)

[9] David, E; Geoffrey, E.R.J.: Learning representations by back-propagating errors. nature, 323(6088), 533-536. (1986)

[10] Dosovitskiy, A; Beyer, L.: An image is worth 16x16 words: transformers for image recognition at scale. arXiv preprint arXiv:2010.11929. (2021)

[11] Eigen, D.; Fergus, R.: Predicting depth, surface normals and semantic labels with a common multiscale convolutional architecture. In Proceedings of the IEEE international conference on computer vision (pp. 2650-2658). (2015)

[12] Farneback, G.: The stereo problem. Excerpt from PhD thesis, Polynomial (2001)

[13] Godard, C.; Aodha, O.F.M.B.G.: Digging into self-supervised monocular depth estimation. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 3828-3838). (2019)

[14] Hochreiter, S; Schmidhuber, J.: Long short-term memory. Neural computation, 9(8), 1735-1780. (1997)

[15] Johnston, A; Caneiro, G.: Self-supervised monocular trained depth estimation using self-attention and discrete disparity volume. In Proceedings of the ieee/cvf conference on computer vision and pattern recognition (pp. 4756-4765). (2020)

[16] Laina, I.; Rupprecht, C.B.V.: Deeper depth prediction with fully convolutional residual networks. In 2016 Fourth international conference on 3D vision (3DV) (pp. 239-248). IEEE. (2016)

[17] Li, H.; Gordon, A.Z.H.: Unsupervised monocular depth learning in dynamic scenes. arXiv preprint arXiv:2010.16404. (2020)

[18] Lin, T.; Maire, M.B.S.: Microsoft coco: Common objects in context. In European conference on computer vision (pp. 740-755). Springer, Cham. (2015)

[19] Mahjourian, R.; Wicke, M.A.A.: 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 (pp. 5667-5675). (2018)

[20] Menze, M; Geiger, A.: Object scene flow for autonomous vehiclese. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3061-3070). (2015)

[21] Ramachandran, P; Parmar, N.: Stand-alone self-attention in vision models. Stand-alone self-attention in vision models. arXiv preprint arXiv:1906.05909. (2019)

[22] Ranftl, R; Bochkovskiy, A.: Vision transformers for dense prediction. arXiv preprint arXiv:2103.13413. (2021)

[23] Ranftl, R; Lasinger, K.: Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. arXiv preprint arXiv:1907.01341. (2020)

[24] Ren, X.; Bo, L.F.D.: scene labeling: Features and algorithms. In 2012 IEEE Conference on Computer Vision and Pattern Recognition (pp. 2759-2766). IEEE. (2012)

[25] Silberman, N.; Hoiem, D.F.R.: Indoor segmentation and support inference from rgbd images. In European conference on computer vision (pp. 746-760). Springer, Berlin, Heidelberg. (2020)

[26] Taylor, J.; Shotton, J.S.T.: The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation. In 2012 IEEE Conference on Computer Vision and Pattern Recognition (pp. 103-110). IEEE. (2012)

[27] Vaswani, A; Shazeer, N.P.N.: Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008). (2017)

[28] Yang, Z.; Wang, P.W.Y.X.W.N.R.: Every pixel counts: Unsupervised geometry learning with holistic 3d motion understanding. In Proceedings of the European Conference on Computer Vision (ECCV) Workshops (pp. 0-0). (2018)

[29] Yuan, Y; Huang, L.: Ocnet: Object context for semantic segmentation. International Journal of Computer Vision, 1-24. (2021)

[30] Yuan, Y; Huang, L.G.J.: Ocnet: Object context for semantic segmentation. International Journal of Computer Vision, 1-24. (2021)

[31] Zhang, H; Goodfellow, I.: Self-attention generative adversarial networks. In International conference on machine learning (pp. 7354-7363). PMLR. (2019)

[32] Zhao, H; Shi, J.: Pyramid scene parsing network. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2881-2890). (2017)

[33] Zhou, T.; Brown, M.S.N.L.D.: Unsupervised learning of depth and ego-motion from video. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1851-1858). (2017)