

SELF-SUPERVISED MONOCULAR DEPTH ESTIMATION FOR DYNAMIC SCENES

SJÄLVÖVERVAKAD MONOKULÄR DJUPESTIMERING
FRÅN DYNAMISKA SCENER

JONATHAN LINDBERG

Master's thesis
2021:E35



LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Master's Theses in Mathematical Sciences 2021:E35

ISSN 1404-6342

LUTFMA-3452-2021

Mathematics

Centre for Mathematical Sciences

Lund University

Box 118, SE-221 00 Lund, Sweden

<http://www.maths.lth.se/>

Self-supervised monocular depth estimation for dynamic scenes

Master thesis - Jonathan Lindberg

Supervisors : Amer Mustajbasic (Volvo Cars) Anders Heyden (LTH)

Abstract

Estimating depth from an image is an ill-fitted problem, since we project a three-dimensional space to a two-dimensional image. However, human can estimate a plausible depth from only a single eye and often are we relying on this for quick estimation of moving objects. Although machine learning has shown to be powerful in computer vision tasks, the process of constructing deep learning models is often connected to collecting large amount of meaningful data. Acquiring per-pixel depth data for different scenes is expensive, tedious and sparse, so a naive approach to use regression or supervised models does not scale when using data-driven development.

Self-supervised depth estimation from monocular data try to instead use only data from a camera stream to estimate depth by using view-synthesis and change the problem of regression to an image reconstruction task. The image reconstruction will then lead indirectly to a per-pixel depth estimation from a single image. This approach is great for a number of various reasons but relies on several different assumptions. One of those is a static scene assumption, which often fails on real life data that is collected from a video camera mounted on a moving car in dynamic scenes. In this thesis, the goal is to investigate self-supervised monocular depth approaches in dynamic scenes. Different methods are applied and evaluated that address the inaccuracies in depth estimation that moving objects causes.

State-of-the-art self-supervised monocular depth approaches studied in this thesis consists of handling the depth estimation through view-synthesis. This create a reliance on correct pixel correspondence from one frame to another for the network to accurately estimate the depth. Motion of objects in dynamic scenes causes the pixel mapping to be incorrect which affects the accuracy of depth estimation in those areas. The best way to handle this was found out to be to remove the pixels where motion has happened from the deep learning network's weights update scheme. This causes the network to not punish accurate depth estimation even though the pixel mapping was incorrect. The most efficient way to find problematic areas was found out to be with a forward-backward optical flow consistency check.

Keywords: Self-supervised learning, monocular depth inference, autonomous driving, optical flow

Acknowledgements

Throughout the project, that will mark the end of my master's program, I have received extensive help and support from my supervisor Amer Mustajbasic at Volvo Cars. We have converse almost daily and he has shown a keen interest in my problems during the project. He has aided me with many things, from trivial questions to fine-tuning the report. I would also like to show gratitude towards Anders Heyden, my supervisor at LTH for helping me with regular meetings and feedback on the report.

Lastly, I would like to thank all my friends and family members that helped me, during and all the years leading up to the thesis.

Contents

1	Introduction	1
1.1	Background	1
1.2	Goals & Purpose	1
2	Theory	1
2.1	Camera model	2
2.2	Depth estimation	3
2.3	Machine learning	4
3	Neural network	7
3.1	Convolutional neural networks	7
3.2	ResNet	8
3.3	U-Net	9
3.4	FlowNet	10
3.5	Transfer learning	10
3.6	Bayesian networks	11
4	Self-supervised monocular depth estimation	11
4.1	Framework	11
4.2	Problems and improvement	13
4.2.1	Depth/Disparity representation	13
4.2.2	Blurry depth map and scale ambiguity	13
4.2.3	Bad matching	14
4.2.4	Dynamic scenes	15
5	Model selection	17
5.1	Monodepth2	17
5.2	MotionNet	19
5.3	Manydepth	19
6	Experiments	20
6.1	Dataset	20
6.2	Data augmentation	21
6.3	Implementations details	21
7	Results	21
8	Discussion	24
9	Conclusion	26

1 Introduction

1.1 Background

Since the start of using convolutional layers in neural networks, the deep learning models has been used for various computer vision tasks. In some cases, the models outperform the traditional methods and in other it can handle tasks that were previously challenging. Latest research in structure from motion (SfM) has introduced a pipeline for deep learning models to estimate depth from a monocular video which previously was challenging. However, depth estimation from a single camera view has a lot of obstacles to overcome. One of those problem is the assumption of a static world, which is not the case in urban settings. The proposed pipeline tries to find the best depth that makes the appearance between two camera frames similar. If there exist moving objects between those two frames, the appearance consistency will not equal an accurate depth estimation which will cause a degraded training. This thesis will therefore study how to estimate depth from only a single camera frame and loosening the assumption of static scenes by handling cases when those are broken, mainly in dynamic environment.

The study and research on self-supervised depth estimation is fairly recent and is moving quickly. The models evolve and fine-tunes to encompass assumptions and is at a stage to be used in real applications. Historically in autonomous driving, sensors such as LIDAR and ultrasound has been used for estimating the three-dimensional scene surrounding the car, but the self-supervised camera models have many advantages and are able to reach similar metric scores. The benefit of using a camera instead of a LIDAR is mainly the lower cost and per-pixel estimation instead of the sparse LIDAR data. Now with the self-supervised method it is also considerably easier to collect data, since no ground truth (depth) is needed for training.

1.2 Goals & Purpose

The thesis primary goal is to investigate self-supervised neural networks for monocular depth estimation in dynamic scenes and evaluate and apply different methods that address the inaccurate depth estimation for moving objects. Self-supervised monocular depth estimation finds its usage in the automobile industry, which wants to use the estimation in many different scenarios. Depth is an important part of the car's perception and is also linked to other part of the car's system. Thus, the depth inference has to be robust and not infringe on other part of the car's ecosystem. Therefore, the thesis will only explore options that uses data from a single video stream. This thesis will aim on investigating the effect of dynamic scenes and how to integrate that into the self-supervised monodepth training pipeline to improve depth estimation. The goal of this thesis is thus to answer these two questions:

- In what way does moving objects from dynamic scene affect the self-supervised framework?
- Can the problems that moving object causes be answered and incorporated into the self-supervised model?

2 Theory

Computer vision takes on a heavy duty to try to mimic or replicate human vision on to something that is comprehensible for the computer. That is to say, the ability to find structural features and other information that is present in a scene with help of one or more cameras connected to a computer. This concept has many applications within e.g. robotics, medicine and other areas where images in any form and kind are present. It is therefore one of the central problems in applied Artificial Intelligence, an area where complexity of human vision is brought a step closer to the machines. We know that from a given scene our eyes and brain can interpret, things like depth or movement, and we thus use the same model i.e. a camera as the sensing device and a computer for the interpretation. Therefore, we have to design a mathematics model for the camera and the interpreting device.

2.1 Camera model

The standard model to describe how objects or scenes in the three-dimensional space is projected to an image is inspired from how early cameras were constructed. It is called the pinhole camera model and is constructed by imagining a wall behind a barrier with a very small hole, the pinhole. When light rays reflect from an object the rays pass through the pinhole and form a picture on the wall. If we set the pinhole, the focal point of the rays, at the camera's origin C we get an image that is upside down. By letting the focal length $f = 1$ and placing the image plane in front of the pinhole we get a plane at $z = 1$ that is a virtual image plane with a non-mirrored rendering. An object $O = (X, Y, Z)$ in the global scene will be projected on the image plane $o = (x, y, 1)$ where the line intersects the image. Since the line intersects with the focal point C we can describe the ray with the parameterisation λ

$$C + \lambda O, \lambda \in \mathbb{R}$$

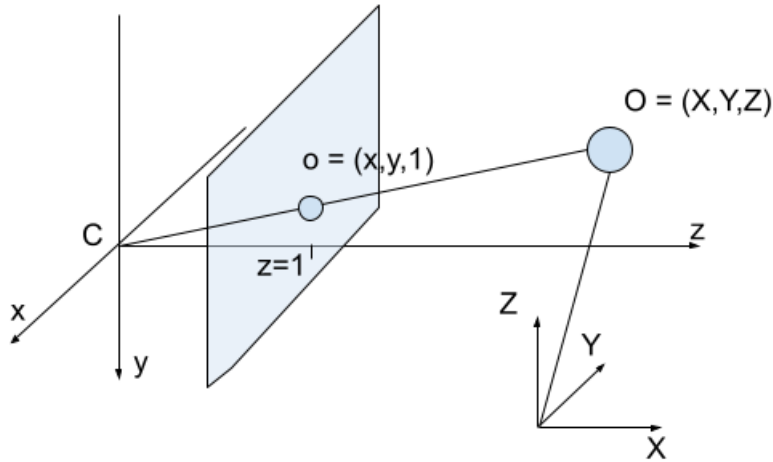


Figure 1: The pinhole camera model, an object O in the global coordinate space (X, Y, Z) will project onto the virtual image plane o in camera coordinates $(x, y, 1)$

The intersection with the plane will occur at $z = 1$, so assuming $Z \neq 0$, from the equation $C + \lambda O = o$ the parameterisation is equal to $\lambda = (1 - C_z)/Z$. Since we set the focal point in the origin, $C = (0, 0, 0)$ we get $\lambda = 1/Z$ and that the mapping from a scene point O to the image plane, in camera coordinates, is

$$\lambda O = \lambda \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} X/Z \\ Y/Z \\ 1 \end{bmatrix} \quad (1)$$

We note that since we project a three-dimensional point to a two-dimensional plane, we lose some information, i.e. the depth since the mapping $o = O/Z$ has infinitely many solutions. We also note that this is a simple model and that we have fixed the camera's focal point and the image is in the camera's coordinate. Therefore we need to map it from the camera coordinates $(x, y, 1)$ to the image plane (u, v) and let the camera's focal length be a variable. This is acquired by constructing a mapping from a general camera plane to the image (u, v) . We do this by the intrinsic camera matrix K

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where the (f_x, f_y) is the camera's focal lengths, (c_x, c_y) is the optical centre and s is the skew. These values are calibrated by different methods and is often given by the camera's manufacture. The mapping from a

scene to the image is thus equation 1 multiplied by the intrinsic camera matrix $(u, v) = \lambda KO$. However, we have still fixed the camera to the global origin. We solve this by moving the camera in the global coordinates system. A rigid camera movement can have 6 degree of freedom, 3 rotations axis and 3 translation. Let $t \in \mathbb{R}^3$ be the translation vector and R be a 3×3 matrix that has the properties $R^T R = I$ and $\det(R) = 1$. To relate the local camera coordinate system and the global coordinate system where the camera and scene live in, we can apply the inverse rotation and translation to the scene, since moving the camera and moving the scene is the same thing. The scene point O is moved by first rotating then translating $O' = RO + t$. If we apply the transformation as above the new image (u, v) at point O' is

$$\lambda O' = \lambda \left[\begin{array}{c|c} K & \mathbf{0}_{3 \times 1} \\ \hline \mathbf{0}_{1 \times 3} & 1 \end{array} \right] \left[\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0}_{1 \times 3} & 1 \end{array} \right] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad (2)$$

where the first matrix, is often annotated as $[R|t]$ and is called the extrinsic matrix/parameters. The camera movement can also be described from an arbitrary position to another by applying the same equation but then the matrix $[R|t]$ is often called the pose.

2.2 Depth estimation

As we saw in the pinhole model, we lose information about the depth. Many objects projects to the same pixels on the image plane, which limits the possibility to retrieve correct depth information. However, if a person looks at a picture, they are able to reason how the scene is constructed in depth, for example the tree is in front of the rock or the car is roughly 10 meters from the camera. We are able to this since our brain can recognise the visual signals in the image, such as occlusion, geometry, texture, motion and are able to easily inference some of the depth information. This does not inherently solve the problem since the brain can get tricked by warped two-dimensional images on a road that appears three-dimensional on the image (optical illusion) and so on. But with reasoning and knowledge, we can assess that objects size close to the camera is larger the farther apart, the texture of, for example grass is of higher quality close and will appear blurry farther away. This is one way we can reason about depth from an image.

Another way that human is able to find the depth of a scene projected on the retina is to have two different projection of the same scene. Which we have, by having two eyes and it is possible to mimic that with two cameras. We can get these two different projections either from a stereo rig taking the image simultaneously or taking two frames sequentially from a moving camera in a static scene.

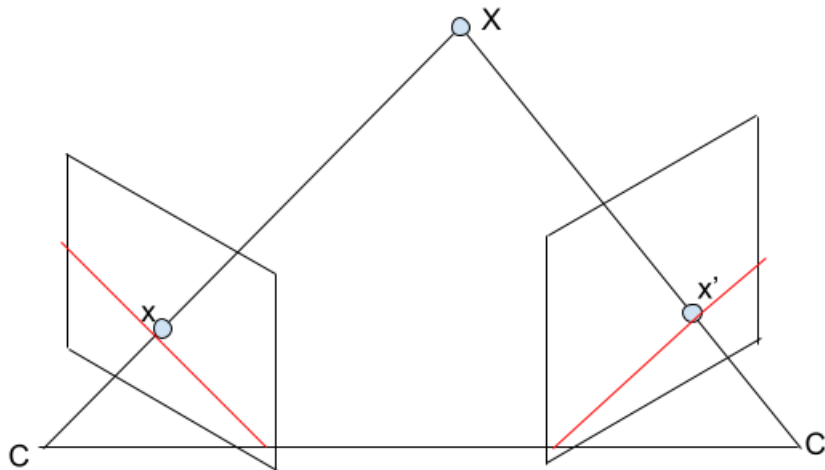


Figure 2: Triangulation with the epipolar lines in red

If we know the intrinsic parameters K and the pose change $[R|t]$ we can find the depth by triangulation as seen in Figure 2. Where the scene point X is where the ray meets and form a triangle from where the object renders on the 2 different images x, x' .

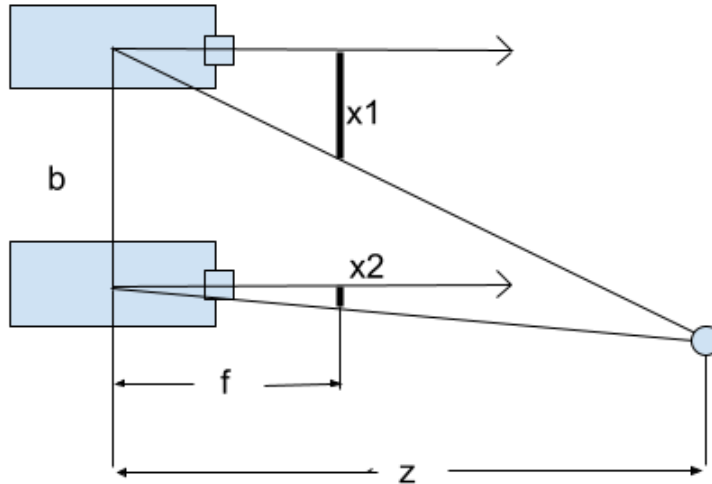


Figure 3: Stereo rig to find the disparity

If we want do the reverse, go from matching feature on x, x' to the depth z we can use similar triangles. We need as above to identify the pixels that matches to each other, this will be simplified by using epipolar lines (in red) to reduce the search to a line. Then, as seen in Figure 3 we can find the disparity i.e. a distance between matching points in image plane $d = x1 - x2$, to get the equation of similar triangles

$$\frac{d}{b} = \frac{f}{z}$$

and then get the depth $z = f \cdot b/d$, where b is the distance between the two camera's centre called the baseline. This is a traditional way to get the disparity map from a stereo rig, where there are many ways to get feature matching and then finding the best mapping from the epipolar constraint. Disparity and depth is often used interchangeably since $z \propto \frac{1}{d}$ and will be used furthermore, depending on the case.

Suppose now that one has a disparity/depth map as well as an image of a scene and want to reproject the projected scene to the world coordinates. One can do this by backpropagation from equation 2, ignoring the camera movement and only focus on the camera coordinates. Let u, v be the camera coordinates and z be the depth found in the depth map at coordinates u, v . Then to get the scene point $O = (X, Y, Z)$ we invert equation 1 and 2

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} X/Z \\ Y/Z \\ Z \end{bmatrix} = \frac{1}{Z} K O \quad (3)$$

$$O = Z K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad (4)$$

2.3 Machine learning

Machine learning is often a catch-all term for algorithms or programs that uses computers to learn without a specific task i.e. not modelled for just a single problem. Many of these techniques are algorithms or artificial

neural networks (ANN, or just nets) that learns from data and then make predictions based on that. A term that has become interchangeable with ANN is deep learning, that is an ANN that several years ago was only 10 layers deep, but is now often 100s of layers deep. One often divides deep learning into three categories, reinforcement learning, supervised learning and unsupervised learning. In reinforcement, the learning is based on interaction with a defined environment and behaviour, while in supervised and unsupervised, one uses data to make inference. Where supervised learning has learned by having a true desired output, while unsupervised only have input data and has to find patterns by defining a task that only uses input data. Self-supervised learning is a mixture of supervised and unsupervised, where there exist no true output data but uses specifically designed tasks as a supervisory signal [10].

Deep learning nets has a rough relation/inspiration from the structure of mammals' brain. A net is constructed by interconnected neurons, that is categorised into layers which exchange information by an activation similar how the brain works. The naive interpretation would then be to just construct a massive interconnected net and train it to output anything, which the mammal brain is capable of doing. The problem is the scale and number of neurons for each interconnected layer explodes with increased connections and the brain has tuned and trained its neurons for over 100 million years [12]. A better approach would be to formulate a structure of different layers with respect to the nets purpose and a way to train specific net by defining a loss suitable to the task being optimised. Then we could construct the general deep learning training problem as

$$\min_{\theta} \sum_{i=1}^N L(m(x_i; \theta) y_i)$$

where $m(x; \theta)$ is a non-descriptive model with training data x divided into batches x_i , $i \in \{1, \dots, N\}$ and y_i is possible true desired data. The learning part is then finding the optimal parameters θ with respect to a loss function $L(u)$ that optimise the intended task.

The model $m(x; \theta)$ usually describes a collection of connections between neurons in a layer with an affine transformation $Wx + b$, where W and b are weights and biases for each neuron. The model also holds the activation functions σ , which is often non-linear, that is applied after each layer to introduce non-linearity to the net.

$$m(x; \theta) = W_n \sigma_{n-1} (W_{n-1} \sigma_{n-2} (\dots (W_2 \sigma_1 (W_1 x + b_1) + b_2) \dots) + b_{n-1}) + b_n$$

The connection is not always an affine transformation and the model is often in practice described with a directed graph. This graph and model description is used to argue for the universal approximation theorem, which state that a layer is able to represent most functions (albeit not generalisable or too large to construct). The graph also let us construct a computational graph which help us with the optimisation with calculating the gradient effectively by backpropagation, which we can then use to find optimal θ with a gradient-based method.

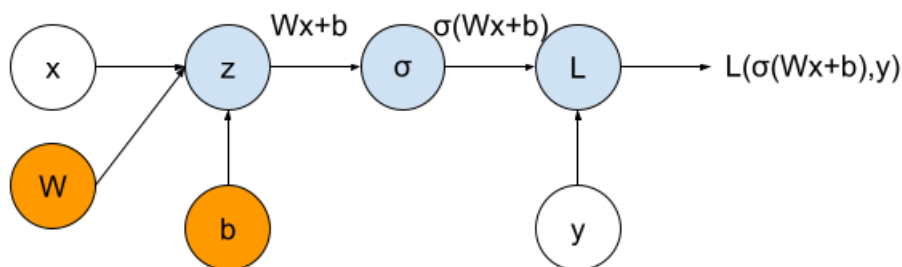


Figure 4: Example of a simple net with a single layer, where the orange nodes is trainable variable

Figure 4 is an example of a graph with a single layer with activation function σ and loss function $L(u)$. The trainable variable is coloured by orange and the input data is in white. This computational graph is

often more complex and can be created dynamically when the code runs. With the computational graph the gradient is easy and fast to calculate by backpropagation. Backpropagation, or reverse-mode automatic differentiation which it is called in other application is a method that uses the chain rule repeatedly and uses the stored information from the forward pass to get the gradient for each trainable variable. The model $m(x; \theta)$ or the linearised graph can be described by its composition of layers as function f_i

$$m(x) = (f_1 \circ f_2 \circ \dots \circ f_n)(x) = f_1(f_2(\dots f_n(x)))$$

then

$$\begin{aligned} \frac{dm(x)}{dx} &= \frac{df_n(z_{n-1})}{dz_{n-1}} \frac{df_{n-1}(z_{n-2})}{dz_{n-2}} \dots \frac{df_2(z_1)}{dz_1} \frac{df_1(x)}{dx} \\ &= \frac{dz_n}{dz_{n-1}} \frac{dz_{n-1}}{dz_{n-2}} \dots \frac{dz_2}{dz_1} \frac{dz_1}{dx} \end{aligned}$$

as $z_n = f_n(z_{n-1})$ is the intermediate values, which becomes known in the forward pass, we are able to effectively get the gradient by multiplying the differential between each layer. This also works in higher dimension and has smart and efficient backpropagation algorithm. So, to train a network is to generate a graph and implement the "local" derivative of an activation function and use the forward and backwards pass to calculate the gradient.

Gradient decent is a great method to find the local minimum in smooth nice low dimensional function, where the optimal value is updated by the gradient and a learning rate γ

$$\theta_{n+1} = \theta_n - \gamma \nabla f(\theta_n)$$

To combat the calculation of the expensive higher dimension Jacobian and reducing the chance to get stuck in a local minimum, a stochastic based gradient decent is often used. One frequently used optimisation algorithm is the ADAM method, that uses the first and second moments of the gradient and step size annealing to find the optimal θ^* . It updates its value by

$$\begin{aligned} m_t &= \beta_1 m_t + (1 - \beta_1) \frac{\partial f(\theta_{t-1})}{\partial \theta_t} \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) \left(\frac{\partial f(\theta_{t-1})}{\partial \theta_t} \right)^2 \\ \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \\ \theta_t &= \theta_t - \gamma \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \end{aligned}$$

where γ is the learning rate, $\beta_{1,2}$ is the decay rate parameters and ϵ is a small scalar to evade division with zero. ADAM works well for deep learning problems with large dataset and/or with a lot of parameters. It is robust and works good at an extended list of non-convex network models and is therefore used in different problems [16].

Often one train the network with a dataset for a set number of epochs and trains until it converges or until it starts to overfit the training data and starts to worsen its prediction for out-of-sample data. To combat the overfitting one can watch when the out-of-sample prediction error starts to grow and stop the training before it happens. This is called early stopping and has been shown to be a good way to combat overfitting that can happen when training a net. Another way is to add regularisation on the weights θ so it punishes complex overfitted models over more simple and general models. In some problem the loss is not significant enough to monitor how the net performs on seen and unseen data. Therefore, one might need some metric to tell if the net performs well or not. The metrics is not used in training but can show how

well the model is doing. Example of metrics, that will be used, is

$$\begin{aligned}\text{RMSE} &= \sqrt{\text{E}[(y - \hat{y})^2]} \\ \text{RMSE log} &= \sqrt{\text{E}[(\log(y) - \log(\hat{y}))^2]} \\ \text{Abs Rel} &= \text{E} \left[\frac{|y - \hat{y}|}{y} \right] \\ \text{Sq Rel} &= \text{E} \left[\frac{|y - \hat{y}|^2}{y} \right] \\ \delta^n &= \text{E} \left[\max\left(\frac{y}{\hat{y}}, \frac{\hat{y}}{y}\right) < \delta^n \right]\end{aligned}$$

These are regression metrics and will measure how close the estimation \hat{y} is to the ground truth y over all data in a batch or over a validation/test set.

3 Neural network

3.1 Convolutional neural networks

Convolutional layered net has historically been used in computer vision tasks but was not as prevalent as they are now. From the 90s until 2010 many tasks were handled by support vector machines, mainly because data and hardware was not there yet and also much of the research was to construct crafted features instead of learned. A lot of improvement in computer vision was in the engineering of features until the famous net AlexNet was proposed. They took another approach and thought that learned features would be the better way to handle a lot of different data and problems. To do so, they put forward that the features should be hierarchically constructed with multiple learned layers both jointly and in different scale. So, for the lower layer, the first filter, will learn edge detection, colour etc. that traditional filter also usually used and higher layers would learn higher level features (more abstract to decipher) such as for example identifying eyes or uncategorisable features. In 2012 AlexNet won the ImageNet challenge by a large margin and thus started a new trend in learning features instead. Its 8-layered architecture is a standard for modern convolutional neural nets and is a sequential net as seen in Figure 5.

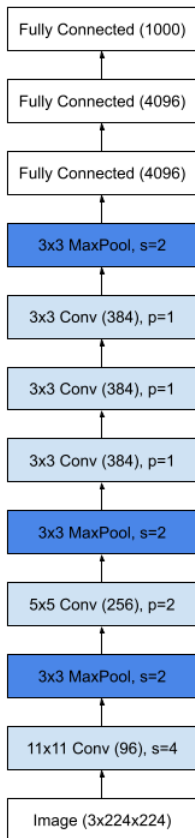


Figure 5: AlexNet architecture, where the spatial dimension is pushed down in the depth dimension and have various different learnable filters

The convolutional layer is a layer with learnable filter that applies the convolutional operator that maps the input (H, W, C_{in}) to $(H_{out}, W_{out}, C_{out})$. We often have to predefine the dimension of the filter/kernel, as in for example Figure 5 where the first layer has trainable 11×11 kernels. This will cause a spatial dimension loss if we don't pad or increase the kernel stride length through the image. In some cases, one wants to keep the dimension and in some, one wants to decrease the features map. One way is to apply a pooling, so one gradually reduces and push the information to a coarser map. Pooling is also a method to reduce the locality that the convolutional layer produce.

Another important addition to convolutional net was the move away from the traditionally used sigmoid activation. When the dimension of layers used increases, vanishing gradient becomes a problem. As we have seen, the network parameters are updated by the calculated gradient that the backpropagation generates. Since the sigmoid function is $\sigma(x) = 1/(1 + e^{-x})$, the value saturates to 1 or 0 for large respective small values. This makes the gradient for such value small and since we estimate the gradient by backpropagation, the gradient vanishes for net with many layers. A solution to this is the ReLu activation $\sigma(x) = \max(0, x)$, that is still a non-linear activation but behaves like a linear for $x > 0$. It thus helps the vanishing gradient problem and also makes layer able to produce sparse outputs.

3.2 ResNet

ResNet shares a lot of similarity to AlexNet and other modern convolutional neural network (CNN). But as the net becomes more complex and deeper, the result does not follow even though the net can handle more complex models. This has many different reasons and non-nested function approximation is one of them. Let \mathcal{F} be the class of functions that our net can produce. We can reach different functions f by different weights and biases etc. and let f^* be the true optimal function. We find the best function that the net can

produce by finding the $f \in \mathcal{F}$ that is closest to f^* , since f^* is probably not reachable in our \mathcal{F} . Since f^* is not reachable we construct a new bigger net that has a new class \mathcal{F}_1 . However, we are not guaranteed to get a better approximation if the new class of function does not contain the smaller class $\mathcal{F} \not\subseteq \mathcal{F}_1$. A way to increase the number of layer and get deeper net without getting further away from f^* is to add layer that is able to construct the identity function $f(x) = x$, this way the new class of function will always contain the old class of functions. This is the idea that leads to ResNet, that every deeper layer should more easily be able to construct the identity function. Assume that the net wants to construct a function from a layer $f(x) := \sigma(Wx + b)$, if we instead add a skip connection so the input is added to the functions output $f(x) + x$, we can instead learn the residual function $g(x) = f(x) - x$. So, if any additional layer hurt the architecture then we can easy skip the layer, other networks such as highway networks and LSTM uses similar mechanics to control the information that each layer additionally add.

ResNet consists of these block with skips as is shown in Figure 6 that is able to be added on each other to increase the depth without inherently disturbing the network.

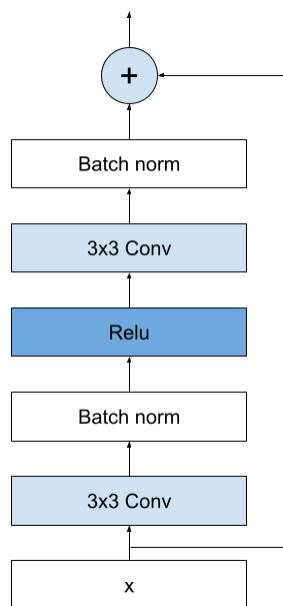


Figure 6: A ResNet block, that has a skip connection and a sequential stacked layers of convolutional layers, batch normalisation and ReLu activation.

The ResNet block contains an additional layer that is called batch normalisation. It is a normalisation step where the values normalise $(x - \hat{\mu}_B) / \hat{\sigma}_B$ and sometimes with an additional scale and shift parameter. The mean and variance are estimated for each batch. This helps the net converge and also the estimation of mean and variance add noise which has in general caused less overfitting and better result. Since no pooling or stride is applied the spatial dimension is reduced, but the original net is designed so the kernel dimension is increased with every block so the net effectively encodes an image (224x224x3) to an encoded space (1x1x512) that can then, depending on the problem, be decoded.

3.3 U-Net

As we saw with AlexNet using fully connected layers with softmax activation good results were achieved when the problem was of a classification type. If we instead want to reconstruct an image, we would like to decode it from the encoded space. This is often achieved by upsampling and adding information from the encoded features or deconvolutional layer to increase the spatial dimension. The features from the encoded space are implemented by adding an additional skip layer for each residual block to another block of equal size. This is often described as a U-net architecture, which was at first mainly used for segmentation in biomedical

images. As previously mentioned in classification problem, usually only the output is used sequentially but when pixel value is needed, often learned features on different level are needed. This is where the name U-net comes from, since the encoding down to lower dimension then decoding it forms a U. The main contribution is that beside upscaling an additional convolutional layer is applied jointly on upscaled feature map and encoder feature map of the same resolution. This let the network learn representation from deep abstract features as well as local information that comes from the convolution.

3.4 FlowNet

A concept that is heavily linked to depth is optical flow, which describes the relative pixel motion between 2 frames. It is a vector field that maps where each pixel changes in u, v from its current position. Traditional way to calculate optical flow is often Lucas–Kanade method, where one assumes a local consistence and solve the differential equation for change in light/colour intensity. This fails when the assumption of small local constant light change doesn't correspond with a moving object. Accomplishment in optical flow measurement from 2 images has been made from various new nets. They are able to do end-to-end estimation and use feature extraction, similar to other CNN, independently and then use spatial pyramids and produce representation between those features to predict flow. FlowNet or a modified FlowNet is shown in Figure 7 where two images are concatenated and encoded to a space which predict a pose $[R|t]$ between the images then upsample the pose to a residual flow in the same resolution as the original images. This works for optical flow and for scene flow i.e. three-dimensions if the images are with depth, RGB-D.

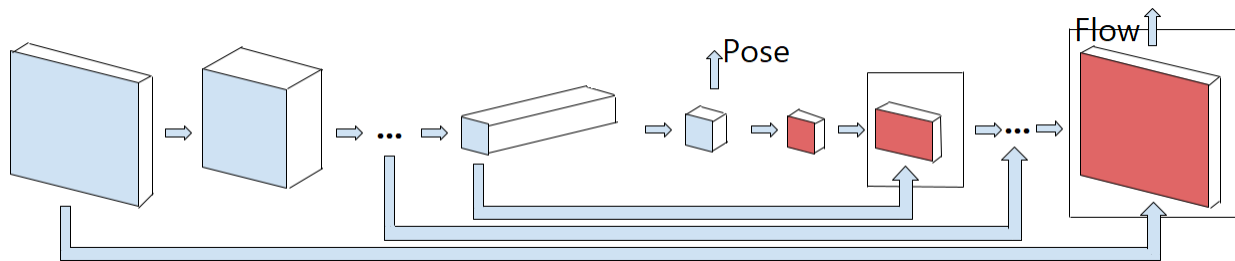


Figure 7: A modified FlowNet, that has a U-Net shape that encodes down the global transformation and then decodes the per-pixel residual translation from various stacked FlowNet blocks.

The flow is decoded as a U-net but have a spatial pyramid block that are able to predicts the global transformation $[R|t]$ and a residual flow.

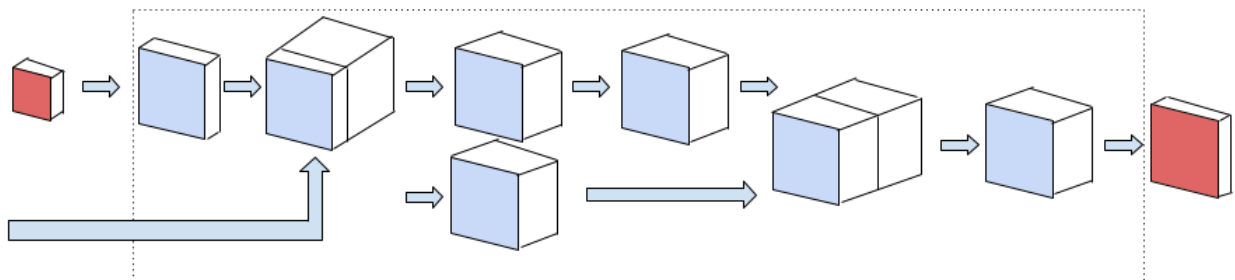


Figure 8: FlowNet blocks that upsample the previous estimated flow with spatial pyramid blocks and skip connections. The arrow represents either convolution, upsampling or concatenation

3.5 Transfer learning

Transfer learning has been described as having the knowledge about how to play the piano helps with learning how to play the guitar. These tasks have some relation for example music scale and those extraction could

be used to both improve a new thing to learn or be a base for something new. In computer vision task some knowledge carries over for different problems such as classification, segmentation and detection. Additionally, complex problem has often an extra demand on the amount of data but there exists some net that has been trained on large dataset, for example the ImageNet. Previously mentioned ResNet is a net trained on the ImageNet dataset for classification of various classes and its weights can be used to produce low level features and some high-level abstraction that is common for all computer vision task. For problem that is not shared with what the net is trained for, the transfer learning can still be used by stripping the top layers to get to the encoded feature space.

3.6 Bayesian networks

Model predictions uncertainty is unavoidable since there always exist some inherent noise in almost all data. Thus, data-driven models such as neural net could benefit from adding modelling of uncertainty into their framework. When speaking of uncertainty that one can model, one usually speaks of two different uncertainty, aleatoric and epistemic. Aleatoric uncertainty is the one usually mentions when talking about uncertainty and is the statistical uncertainty that captures the noise from the data. Epistemic is then instead the uncertainty that captures the uncertainty in the model.

In computer vision there exist techniques that model uncertainty for visual task such as particle filtering, but for deep learning the approaches has not been as developed. For regression and segmentation term, Bayesian tools has been used to construct new losses and nets to capture the two uncertainties [14]. One can encapsulate the epistemic uncertainty by putting a prior distribution on the net’s weights. The model can then be seen as a Bayesian neural network (BNN) and is easy to formulate but often hard to make inference on, since the posterior is need and often hard to acquire. Variational Dropout can be used as a technique to sample the posterior with help of Monte Carlo simulation. This is done by having a dropout layer before the weight layer with a prior on it. Dropout layer is often used as a regulariser in different net and is a simple layer that during training randomly sets the input to a neuron to 0. In true BNN the dropout is kept on during test time and depending on the problem is sampled from to get a variance from the network’s predictions. If we have set Gaussian prior on the weights and for regression problem set our likelihood also to a Gaussian with the models output as the mean, we get

$$p(y|f^W(x)) = \mathcal{N}(f^W(x), \sigma^2)$$

where $f^W(x)$ is the output from the BNN with weights W . The predicted variance for a regression is then

$$\text{Var}(\hat{y}) \approx \sigma^2 + \frac{1}{T} \sum_{t=1}^T f_t^{\widehat{W}}(x)^T f_t^{\widehat{W}}(x_t) - \left(\frac{1}{T} \sum_{t=1}^T f_t^{\widehat{W}}(x) \right)^2$$

where \widehat{W} is the sample weight after the dropout layer. The first term σ^2 is the heteroscedastic aleatoric uncertainty. Heteroscedastic aleatoric uncertainty means that the noise σ^2 can vary dependent on the input data x . Then if we also let the net predict σ we get from a net a prediction and a variance $[\hat{y}, \hat{\sigma}^2] = f^{\widehat{W}}(x)$. Then, as we draw model weights W and get the approximated posterior from the network the loss for a regression problem can be constructed such that it optimises for the regression and uncertainty without having any uncertainty ground truth.

$$L_{BNN} = \frac{1}{D} \sum_i \frac{1}{2} \frac{\|y_i - \hat{y}_i\|^2}{\hat{\sigma}_i^2} + \frac{1}{2} \log \hat{\sigma}_i^2 \quad (5)$$

4 Self-supervised monocular depth estimation

4.1 Framework

Since introducing the ImageNet project with its 14 million hand-annotated images [3] it has been possible to improve numerous computer vision task where high quality annotated data is needed. Classification net,

like Google’s EfficientNet, has reached a 90 % top-1 accuracy which has made net usable in the industry and features extracted from such net are used for other visual tasks [24]. However, in depth estimation, annotated data is hard and expensive to find. Annotated depth data is acquired from high-resolution LIDAR that must be calibrated. Self-supervised method is thus often needed in depth estimation, since it is easy to record video mounted on for example a station wagon and there exist many public datasets on recorded street view. The pipeline for self-supervised monocular depth estimation was first proposed by Zhou et al. in 2017 which has been built up on [25]. The input data for the model is RGB images and it uses view-synthesis to indirectly learn the depth in the scene by utilising three-dimensional scene geometry. By using equation 2 and 4, one can construct where the pixels from one image is placed on a new image given depth and camera movement. So if a camera is placed on a car, and we are able to estimate the depth and ego-motion of the car/camera then the image at time $t + 1$ should be able to be reconstructed from time t and vice versa.

Since the information for reconstructing information exist in the video, it might be possible to construct a net that predicts a depth and a net that predicts the ego-motion. This can be done by minimising a proxy photometric loss between an image at $t + n$, where $n \in \mathbb{Z}$ and a warped predicted image from time t . To get a photometric difference a differential pixel value (colour) has to be derived as well, generated from the view-synthesis. This is possible by the (sub-)differentiable sampling mechanism inspired from Spatial Transformer Networks [13]. It is an interpolation method using the backward warping method to get a dense image registration. It does backward warping, instead of forward, since it is easier to interpret backward warping for when the pixel values is warped to a non-integer pixel. It does the sampling by using a sampling kernel which is applied on the colours independently $I_i^c, c \in [1 \dots C]$ on the new pixels (x_i, y_i)

$$\hat{I}_i^c = \sum_n \sum_m I_{nm}^c k(x_i - m; \Phi_x) k(y_i - n; \Phi_y), \quad \forall i \in [1 \dots H'W'] \quad \forall c \in [1 \dots C] \quad (6)$$

where $k(u; \Phi_u)$ is a generic sampling kernel with parameters Φ_u . Usually the traditional bilinear sampler is used $k(u; \Phi_u, m) = \max(0, 1 - |u_i - m|)$

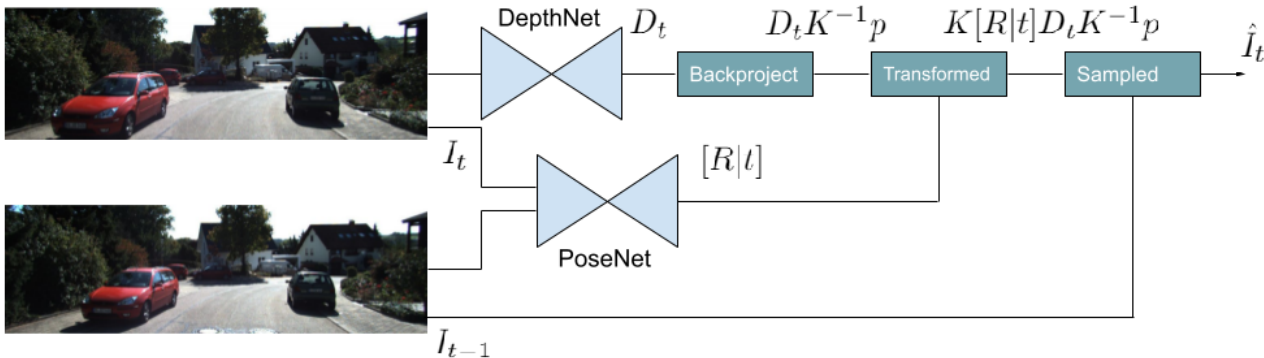


Figure 9: The self-supervised framework, where a source frame depth D_t is predicted and a pose $[R|t]$ is estimated from a source frame and a target frame. The coordinates from the source frame is warped to where, with perfect predicted depth, it will be placed on the correct pixels on the target frame. Then to make it differentiable (since we are in pixel coordinates) we use a bilinear sampler to reconstruct the image from the target image. This will result in a reconstructed image that should look like the source frame.

Figure 9 shows the overall pipeline to construct a view-synthesized image that has been reconstructed from time $t - 1$, which works for any time $t + n, n \in \mathbb{Z}$. The DepthNet for the original framework has a U-Net structure and have a pretrained encoder from ResNet-18 and a decoder with multi-scaled outputs $[HW, H/2W/2, \dots, H/nW/n]$ which is inspired from pyramid representation. The PoseNet also has a ResNet encoder, which can either share the encoder with the DepthNet or be pretrained, following a number of convolution layers. The output of the PoseNet predicts the 6-Dof to form the camera pose $[R|t]$, it is scaled with 0.01 as in [7] (which is the framework this thesis is mainly built on).

When the reconstructed image \hat{I}_t is predicted, the loss is calculated on how similar the true I_t is with the predicted one. The way to measure the similarity between two images is a non-trivial one since there is more to consider than only difference in intensity pixel-wise. Structural similarity index measure (SSIM) takes care of more spatial information, than the per-pixel approach, since there exists dependency in close pixel values. SSIM uses luminance, contrast and structure to estimate the similarity by

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

where $\mu_{x,y}$ is the mean, $\sigma_{x,y,xy}$ is the covariance and $c_{1,2}$ are scalar values.

4.2 Problems and improvement

Reconstruction through view-synthesis provides a minimalistic approach to estimate depth from a single frame but it is conditioned on several assumptions. This opens up for several areas to improve the pipeline.

4.2.1 Depth/Disparity representation

For outdoor scene the depth values are higher than what a convolutional neural net can produce stably, therefore there is a need for a representation from the output. Monodepth2 [7] has a sigmoid activation on the DepthNets outputs feature and instead of treating it as a depth map it treats it as a disparity map. Then they invert it to get the right representation (depth). Better results were achieved when using scaled inversion instead

$$d = \frac{1}{\sigma_{\min} + (\sigma_{\max} - \sigma_{\min}) \cdot x} \quad (7)$$

where $\sigma_{\min} = 0.1$ and $\sigma_{\max} = 100$. MotionNet [19] proposed a new way to learn the depth directly by having a softplus activation instead

$$d = \log(\exp(x) + 1)$$

which is linear for large x and is smooth and avoid the cases when $d = 0$. This representation is also studied in [15] and the representation removes the dependency on hyperparameter and instead let the networks last convolution layer be able to find the optimal representation.

4.2.2 Blurry depth map and scale ambiguity

The current loss works as a proxy for learning depth by working with the three-dimensional geometry and checks purely appearance similarity. This allows us to do self-supervised learning but comes with several problems since a good match with the reconstructed image does not always match with a good depth estimation. A solution to some of these problems is to add additional loss on the disparity map. To handle problem of smooth surfaces and depth transition at edges an additional disparity smoothness loss is added. Thus, instead of having the loss as only a reconstruction loss $L = \text{SSIM}(I_t, \hat{I}_t)$, the loss is a photometric error (pe) and a disparity smoothness loss L_s

$$\begin{aligned} L_{\text{pe}} &= \frac{\alpha}{2}(1 - \text{SSIM}(I_t, \hat{I}_t)) + (1 - \alpha)\|I_t - \hat{I}_t\|_1 \\ L &= L_{\text{pe}} + \lambda_s L_s \end{aligned} \quad (8)$$

where $\alpha = 0.85$, $\lambda_s = 10^{-3}$ is hyperparameters and the smoothness loss is

$$L_s = |\partial_x d_t| e^{-|\partial_z I_t|} + |\partial_y d_t| e^{-|\partial_y I_t|}.$$

It is an edge-aware smoothness that enforces change of disparities to be small on textureless surfaces but large at the edges i.e. at the objects surface the disparity is similar while at the edge the disparity changes. This is enforced by having the gradient to the input image I_t as a supervised guide for edges and smooth surfaces.

As we saw in the camera model that when we project three-dimensional scene objects to the camera frame, we lose a dimension which in turns causes scale ambiguity. That is, the same scene can be produced with different depth estimation but have the same interpolated image. This causes a problem in training since two disparity maps can be the same with respect to a scale, then when optimising the loss from the disparity map, the disparity collapses to a scale of 0 because of optimal smoothness loss occurs when $d \rightarrow 0$. The way to fix this is to apply a non-linear function to normalise the disparity before calculating the loss on it. One choice that [22] propose is to divide the output of the depth net with its mean

$$\bar{d} = \frac{d}{E[d]}$$

So the smoothness loss should be applied on the normalised \bar{d} instead of the outputted network disparity prediction.

4.2.3 Bad matching

For the operation of the monodepth framework to work it relies on features/pixel matching. There are several natural properties that can introduce errors in the feature matching process such as low texture and repeating regions, different illumination and occlusion. Some of these are unavoidable and occurs mostly in various outdoor scenes. Repetitive and textureless region appears for example in the sky as many pixels has the same colour intensity and thus matching each pixel to new pixel is hard. Different illumination causes similar problems and an assumption about the light properties is often presumed. That is that all object follows the Lambertian property i.e. the apparent brightness is consistent independently of the observer’s view. In practice the property is often broken at strongly reflective surfaces such as glass windows, which causes a glare and the depth estimation on the glare doesn’t correspond to the real depth. Some research combats this with learning a simple transformation $a \in \mathbb{R}, b \in \mathbb{R}$ for each different frame so additionally the reconstructed image is $I' = a \cdot I + b$. The SSIM loss already helps a little by working with the structural similarity instead of the pixel similarity. MotionNet [19] expand this loss to comprehend the more severe cases, by comparing the estimated depth and the warped depth. If they are not the same, assuming good depth estimation, then there has been a bad matching caused by occlusion or violation of the Lambertian property. Then in the loss function the SSIM is weighted with the depth error caused by the bad matching

$$w^{(i,j)} = \frac{\sigma_d^2}{\sigma_d^2 + \left(\hat{d}_{t,\text{recon}}^{(i,j)} - \hat{d}_t^{(i,j)}\right)^2} \tag{9}$$

where

$$\sigma_d^2 = \frac{1}{N} \sum_{i,j} \left(\hat{d}_{t,\text{recon}} - \hat{d}_t\right)^2$$

the \hat{d}_t is the estimated depth and $\hat{d}_{t,\text{recon}}$ is the warped reconstructed depth from a nearby frame. The idea behind it is that in a bad matching we scale or mask out the loss on that pixel. As seen in the Figure 10, assuming a stationary camera feed a mismatching is caused by occlusion/motion. Then the warped depth should be equal or higher on the target frame for it to be eligible. This lay ground for a method to handle occlusions and dynamic scenes.

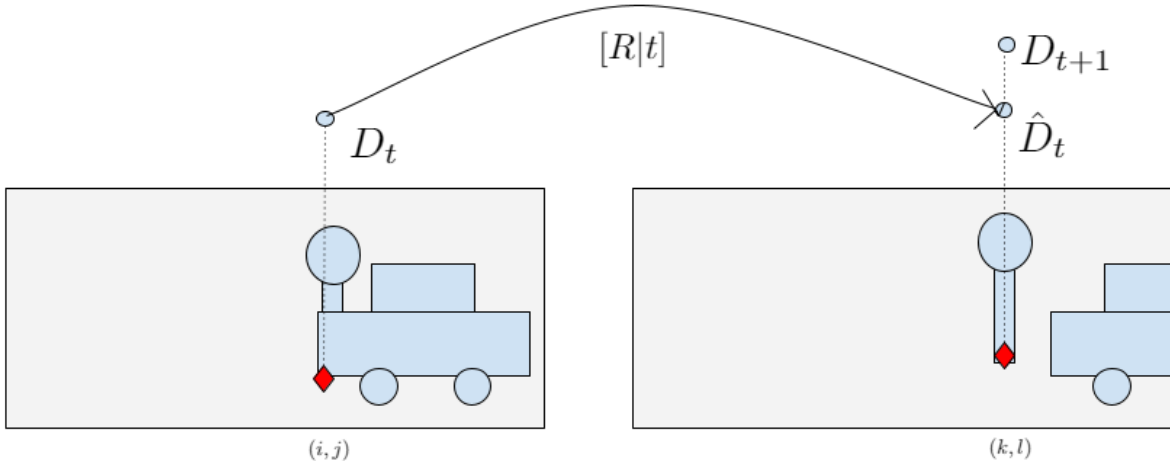


Figure 10: Example of a bad matching that happens because of occlusion/moving objects. In frame t the car is at pixel (i, j) and is in front of the tree (with respect to the camera) and in frame $t + 1$ the view-synthesis pixel is placed on pixels (k, l) with given depth, pose and stationary should be correct. Instead the depth estimated from $t + 1$ sees that the tree is at pixel (k, l) and estimate a depth farther away than the warped depth from frame t , so $\hat{D}_t < D_{t+1}$.

Occlusion happens when an object is shown in one frame but not in another. We would not want to reconstruct an object that is not visible, since there is nothing to compare the reconstruction to. In a monocular video, this could for example happen when the automobile turns or pass some objects, revealing new objects. The primary solution to occlusion is to do two reconstructions, one forward in time and one backwards. Then for each matching pixel choose the matching with the lowest photometric error.

$$L_{occ-pe} = \min_{t'} pe(\mathbf{I}_t, \mathbf{I}_{t' \rightarrow t}) \quad (10)$$

where t' is another frame $t' \neq t$ warped to match frame t . The thought behind it is that the loss should be higher when comparing an unseen occluded object than if comparing non-occluded ones. Other way is to mask out i.e. not calculating the loss on objects that is occluded, since a bad matching will trick the gradient to move away from current estimation even though the depth could have been correct. The masking can depend on many different variables and should remove the pixels when it can't match because of missing correspondence on either the source or target image. An approach could be to use the technique, as seen in Figure 10, to mask out if the reconstructed depth appears in front of the estimated depth. This however requires symmetry i.e. to check the reverse reconstruction.

4.2.4 Dynamic scenes

Moving objects also causes occlusion but also creates a unique problem to the depth estimation. The occlusion part implies a mismatch even though an accurate depth could result in a wrong matching and is therefore punished and degrades the depth estimation task. But the more severe problem happens when there exists moving object that has the same, or roughly equal velocity. This will cause an estimation of infinite depth, as seen in Figure 11. The cause is that the moving object is with relation to the camera fully stationary, so to warp those pixels the disparity will become zero. This is a serious problem for autonomous driving as this situation happens a lot and the consequence of wrongly estimating moving object could be severe.



Figure 11: Example of a infinite depth/ "hole" that is caused by moving objects that has the same velocity has the camera. [7]

Monodepth2 handles this problem with infinite depth estimation by masking out pixels when they appear stationary with relation to the camera. This happens for objects that travels with the same velocity as the camera and when the camera is stationary. The mask is based on that when appearance do not change from frame to frame, the pipeline cannot handle those pixels. So by having a per-pixel mask $\mu \in [0, 1]$ we can remove the calculating of loss on those pixels. Monodepth2 authors propose that no appearance has changed if the photometric loss for only the camera movement is less than the warped reconstruction

$$\mu = \left[\min_{t'} pe(I_t, I_{t' \rightarrow t}) < \min_{t'} pe(I_t, I_{t'}) \right] \quad (11)$$

where $[]$ is 0 respectively 1 when false/true and is called the Iverson bracket. This approach works well for moving objects that appears stationary to the camera but the depth estimation for general moving object will still be calculated wrongly. If we could estimate the moving objects, we could additionally mask out the part of the moving objects that causes occlusion. Optical flow can describe the pattern of discernible motion if we remove the motion cause by the camera movement and then look at residual motion. If we define the flow from the estimated depth as how each pixel p_t changes

$$F_{t,t'}(p_t) = K[R|T]_{t,t'} D_t(p) K^{-1} p_t - p_t. \quad (12)$$

The pixel for static object, assuming good estimation of pose and depth, will be close to zero when the flow forward and backward is applied. We can mask out pixels, like [21] that mask out pixels that does not have a zero forward-backward flow by masking out uniformly, in a circle around the pixel like

$$\begin{aligned} & |F_{t,t'}(x) + F_{t',t}(x + F_{t,t'}(x))|^2 \\ & > \gamma_1 \left(|F_{t,t'}(x)|^2 + |F_{t',t}(x + F_{t,t'}(x))|^2 \right) + \gamma_2 \end{aligned} \quad (13)$$

where γ_1, γ_2 are hyperparameter. Similar to the auto mask in equation 11 we remove the pixels that falls outside a radius around the pixel. Another note is that we could identify areas where local motion as occurred, albeit in the image coordinate and just add those per pixel movement in our transformation. Assuming, instead we can get a three-dimensional per pixel translation for moving object we could change equation 2/4 so the new pixel coordinates is

$$p' = K[R|T + T_{obj}] D_t(p) K^{-1} p = K R K^{-1} D_t p + K(T + T_{obj}) \quad (14)$$

This will allow us to handle all type of local motion and stop the mismatching that happens from moving objects. If we allow T_{obj} to be any local translation we get a high complexity since we allow per-pixel translation $T_{obj} \in \mathbb{R}^{3 \times W \times H}$ and if we want to continue our self-supervision we need some kind of regulation. Say that we construct a new net that predicts the translation for moving objects T_{obj} , we need to apply a loss on it to regularise it enough while not restricting model diversion. MotionNet constructs a loss on the motion translation, by noticing that the translation should be sparse, and the movement is often a constant rigid movement. They propose a $\frac{1}{2}$ -norm and group smoothness

$$\begin{aligned} L_{1/2}[T_{obj}(u, v)] &= 2 \sum_{i \in \{x, y, z\}} \langle |T_i| \rangle \iint \sqrt{1 + |T_i(u, v)| / \langle |T_i| \rangle} du dv \\ L_{g1}[T_{obj}(u, v)] &= \sum_{i \in \{x, y, z\}} \iint \sqrt{(\partial_u T_i(u, v))^2 + (\partial_v T_i(u, v))^2} du dv \end{aligned} \quad (15)$$

where $\langle |T_i| \rangle$ is a spatial average operation on $|T_i(u, v)|$. We can also use the technique as we did with the optical flow mask and require a consistency in both forward and backward direction to regulate and improve camera pose

$$L_{cyc} = \alpha_{cyc} \frac{\|\mathbf{R}_{t,t'}\mathbf{R}_{t',t} - \mathbb{1}\|^2}{\|\mathbf{R}_{t,t'} - \mathbb{1}\|^2 + \|\mathbf{R}_{t',t} - \mathbb{1}\|^2} + \beta_{cyc} \iint \frac{\|\mathbf{R}_{t',t}T_{t,t'}(u, v) + T_{t',t}(u_{warp}, v_{warp})\|^2}{\|T_{t,t'}(u, v)\|^2 + \|T_{t',t}(u_{warp}, v_{warp})\|^2} du dv \quad (16)$$

5 Model selection

To find method and nets that can improve depth estimation on dynamic scene with only data from a single camera, three different nets with different configuration are tested. They are all self-supervised and have roughly the same training framework. The DepthNet is an encoder-decoder U-net with pretrained ResNet-18 weights [11] for the encoder with skip connections and multi-scale predictions where from the feature space it is deconvoluted and upsampled through the decoder. The decoder deconvolute and upsample the features then add the skip connection by concatenating it. It thus gets higher resolution information which is then convoluted to learn upsampled features.

All convolution for the decoder has convolution kernel size of three and has activation function ELU $\sigma(x) = \{x, x > 0 | e^x - 1, x \leq 0\}$. The different multi-scaled prediction is then bilinearly interpolated to the original resolution and is, dependently on the pose estimation, warped as $p' = K[R|T]D_t(p)K^{-1}p$. Then they are sampled as in equation 6 with a bilinear sampling kernel. Intermediate layers are extracted as depth estimates on the coarse features, so the output of the depth net is multi-scaled and upsampled since the gradient locality of the bilinear sampler [13]. Also for low-texture area when predicted on lower dimension causes ‘holes’ in the final depth map. With upsampled depth maps it is possible to keep the high-resolution reconstruction so that each part of the depth net from the decoder works toward the same high-resolution reconstruction. Then the loss on the reconstructed images and smoothness on the predicted disparity map is calculated and backpropagated as in equation 8. This is the fundamental procedure for all different models, but they differ on how they handle motion.

5.1 Monodepth2

Monodepth2 without auto mask (MD2 w/o automask)

This is the baseline Monodepth2 without auto mask and follows the pipeline as above with a sigmoid prediction on disparity transformed to depth as equation 7. It predicts the pose by an encoder-decoder net with pretrained ResNet-18 weights. The decoder consists of four convolutional layers with kernel size of three, followed by a ReLu activation. Since the output dimension is lower than for the depth, we do not have to upsample but just apply convolutional layer and have the last layer a channel size of six without any activation layer and in that way estimate the pose. Following [22] the predicted pose is scaled with 0.01 and the input is the images concatenated with source frame t first following the target frame t' , since the input to the encoder is now of size 6 the pretrained encoder weights are copied for the first layer to expand the input to fit. Finally, the loss is

$$L = L_{occ-pe} + 0.001L_s$$

Monodepth2 with auto mask (MD2 automask)

Auto mask is one way to handle the more severe problem of dynamic scene, i.e. when moving objects appear stationary to camera. This model is the same as previously mention but with addition that it only updates the gradient for pixels that doesn’t appear stationary to the camera according to equation 11. The photometric error of the true images are additionally calculated and a Gaussian noise is added with mean 0 and variance 10^{-5} to break ties. The loss becomes

$$L = \mu L_{occ-pe} + 0.001L_s$$

where μ is the auto mask from equation 11.

Monodepth2 with softplus (MD2 softplus)

A newer approach proposed by [19] is to let the DepthNet learn the depth directly instead of learning a representation in form of a disparity and scaling the output with prior knowledge of scaling parameters. The output from the DepthNet is instead replaced with a softplus activation, which appear linear for large value and smoother out towards 0 for smaller value. This avoids the case where the depth can be 0 which causes problem in the warping phase. However, the network sometimes diverges and becomes unstable with the softplus representation, so to avoid this an extra loss is added to penalise when the depth is estimated as a constant. The variance of the depth divided by its mean is punished since the variance of $d/E[d]$ is low when the depth appears constant.

$$L_{var} = 1/\text{Var} \left[\frac{d}{E[d]} \right] = 1/E \left[\left(\frac{d}{E[d]} - E \left[\frac{d}{E[d]} \right] \right)^2 \right] = 1/E \left[\left(\frac{d}{E[d]} - 1 \right)^2 \right]$$

The net is otherwise the same as Monodepth2 with auto mask but only a single-scale prediction is necessary which speeds up the training and the total loss is

$$L = \mu L_{\text{occ-pe}} + 0.001 L_s + 10^{-6} L_{var}$$

Monodepth2 with flowmask (MD2 flowmask)

The forward-backward consistency could identify pixels that is mismatched incorrectly. By having two independent depth estimation we can warp the pixels to the other image and then warp the interpolated pixel values back with the new depth estimated. This is similar to how the MotionNet handle occlusions with its depth inconsistency weighting. If we mask out pixels by estimating the relative optical flow like equation 12, with hyperparameter $\gamma_1 = 0.01, \gamma_2 = 0.5$ or with just a flat out radius that is calculated from the mean of the forward-backward flow, we could remove mismatching from occlusion and motion. If we set μ_{flow} to be when equation 13 is true/false, the loss becomes

$$L = \mu_{flow} \mu L_{\text{occ-pe}} + 0.001 L_s + 10^{-6} L_{var}$$

Monodepth2 with Huber loss (MD2 huber)

The mismatching that motion of objects, outside of those that has the same velocity, causes can be seen as a noise/outlier. In the photometric loss we use a robust loss L1, due the nature of the problem and its reduced sensitivity for large error. Barron [1] put forward a family of functions and a way to automatically determine the suitable robustness. Through constructing a function that can be viewed as a negative log likelihood of a probability distribution and have the robustness as a latent variable, then the maximisation let the net control the robustness. Let $\alpha \in \mathbb{R}$ and $c > 0$, the adaptive loss function is

$$l(x, \alpha, c) = \frac{|\alpha - 2|}{\alpha} \left(\left(\frac{(x/c)^2}{|\alpha - 2|} + 1 \right)^{\alpha/2} - 1 \right)$$

where $\alpha = 1$ the function is a smooth L1

$$l(x, 1, c) = \sqrt{(x/c)^2 + 1} - 1$$

and when $\alpha = 2$ it is undefined but behaves like a L2 loss on the limit

$$\lim_{\alpha \rightarrow 2} l(x, \alpha, c) = \frac{1}{2} (x/c)^2$$

A variation of this inspired by [4] was used since it behaves like traditionally L1 for large error but punish small error similar to how L2 works

$$L_{huber} = \begin{cases} \frac{(I - \hat{I})^2 + c^2}{2c} & \text{if } |I - \hat{I}| \leq c \\ |I - \hat{I}| & \text{if } |I - \hat{I}| > c \end{cases} \quad (17)$$

where $c = \frac{1}{5} \max |I - \hat{I}|$ is the pixels that has a 0.2 less absolute difference $|I - \hat{I}|$ then maximum over all the pixels. The new loss is then the adjusted photometric loss, equation 8, when now the L1 loss is changed to L_{huber} and the total becomes

$$L = \mu L_{occ-huber} + 0.001 L_s + 10^{-6} L_{var}$$

Monodepth2 with uncertainty (MD2 uncertainty)

To capture the aleatoric uncertainty we add an extra head to the depth decoder that is a convolutional layer without any activation function. Since we are training fully self-supervised, we lack the true depth, however [17] showed that the regression task can carry over to the reconstruction task from the photometric error

$$L_{BNN}(\theta) = \frac{L_{occ-pe}}{\hat{\sigma}_i^2} + \log \hat{\sigma}_i^2$$

Training directly causes instability and therefore the output of the aleatoric header should be trained to estimate the log-variance instead. This causes the variance to be non-zero and helps with the stability. The log likelihood loss becomes

$$L_{alea} = e^{-u} L_{occ-pe} + u$$

where $u = \log(\sigma)$ is the output of the aleatoric head. We notice that σ resemble the scaling threshold in losses as Huber but with this method the model itself train a data-dependent "threshold". The total loss is then

$$L = \mu L_{alea} + 0.001 L_s + 10^{-6} L_{var}$$

5.2 MotionNet

MotionNet (MotionNet)

This net differs a bit from the Monodepth2 fundamental structure. The depth network is the same with a softplus representation for the depth but has an additional randomised layer normalisation [8] instead of batch normalisation. According to the authors during training with batch normalisation the long-term means and variance was that of a randomised layer normalisation, which only differs to batch normalisation in that it applies element-wise scale and bias instead for the entire batch. The pose network is a form of FlowNet instead where the input consists of concatenated input frames along with their estimated depth maps from the DepthNet. The FlowNet latent space is convoluted with 2 1x1 convolution layer to predict the pose and is then progressively upsampling features as seen in Figure 8 to output a motion map $T_{obj}(u, v)$ in the original resolution. The motion map is then added to the global translation so the view-synthesis step becomes like equation 14 and then sampled to get the reconstructed frame. The MotionNet is trained symmetrically with 2 frames t, t' , where the symmetry comes from the swapping of order to t', t . The depth is however trained independently and only one forward pass for the images is applied. The MotionNet has no auto mask nor the occlusion aware minimisation method, instead they let the motion map handle all the moving objects and the occlusions are handled from the weighted SSIM loss, described in equation 9 and with the depth mask on the L1 reconstruction error. The loss is

$$L = 3 \times \frac{1 - \text{SSIM}_{weighted}(I_t, \hat{I}_t)}{2} + \|I_t - \hat{I}_t\|_{1, d > \hat{d}} + 0.2 L_{1/2} + L_{g1} + L_{cyc} + 0.001 L_s + 10^{-6} L_{var}$$

where the motion map T_{obj} is normalised by the total translation $\frac{T_{obj}}{\sqrt{3} \|T_{obj} + T\|_2}$ and the hyperparameter for the cycle consistency is $\alpha_{cyc} = 0.001, \beta_{cyc} = 0.05$.

5.3 Manydepth

Manydepth (Manydepth)

A big downside with monocular depth estimation in test time is that for a video sequence the estimation is not perfectly consistent between frames. This is practically not a problem, however there exist potentially more information in a video sequence then camera frames that could be used in test time. Recently an approach

build on Monodepth2 was constructed by [23] that uses temporal information, if able, to estimate depth. It uses a trend in SfM networks called 3D cost volume that helps the network with structural constraint. In Manydepth the depth net takes in nearby frames and with estimation of pose from each to the main frame the pixels can align and a depth estimation in feature representation can then be stacked on top of each other in order and be seen as a binned estimation of a depth distribution for that pixel. The cost volume is then used with the encoded features to decode into multi-scaled disparity maps. The cost volume however let the network become over-reliant on it and error from moving objects and low-texture regions becomes amplified. They thus propose a student-teacher approach where the teacher is the standard pretrained Monodepth2 and gradient towards it are blocked. The depth prediction $d_{teacher}$ is then used to create a mask where the student is over-reliant on wrongly estimated pixels and let the student approach the teacher’s estimation from those pixels. If d_{min} is the smallest depth value that the cost volume predicts then the mask when consistency between teacher and student should be applied is

$$M = \max\left(\frac{d_{min} - d_{teacher}}{d_{teacher}}, \frac{d_{teacher} - d_{min}}{d_{min}}\right) > 1$$

and for those pixels an L1 loss is applied

$$L_{depth-consistency} = \sum M|d - d_{teacher}|$$

so, the final loss is

$$L = (1 - M)\mu L_{occ-pe} + L_{depth-consistency} + 0.001L_s$$

6 Experiments

6.1 Dataset

The data used for training and testing all net was collected from a station wagon, sketch seen in Figure 12, called the Kitti Vision Benchmark Suite [6]. The station wagon was equipped with 4 (2 grey, 2 colour) 1.4 Megapixels cameras with known intrinsic values and the relation to each other and to the car was also known. Additionally to the cameras, the car had a Velodyne HDL-64E 360° field of view LIDAR sensor. The Kitti dataset is one of the most used datasets for autonomous driving tasks since it has both camera input but also LIDAR data as a validation reference. The dataset consists of captured videos recording from a car driving around German cities during various days and different roads. The videos were mostly captured during the day for both static and dynamic scenes and had up to 15 cars and 30 pedestrians visible at the same time.

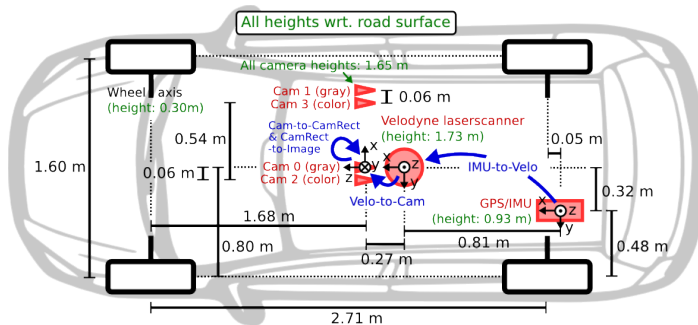


Figure 12: A schematic of the used station wagon for the KITTI dataset [5]

The 2 colour cameras were handled independently of each other, as the stereo rig was not used under training nor under testing. The training data was collected into frames of triplets i.e. frames from $(t - 1, t, t + 1)$ and the training set, following the standard [25] proposed, was sliced into training 39,810 triplets and 4182 test single frames. From those test frames, 100 static scenes and 100 dynamic frames was picked out for testing depth estimation specifically on static respectively dynamic scenes. For evaluation, the prediction was scaled

and cropped to fit the LIDAR data, the prediction was from the left colour camera and was warped to the LIDAR’s position to calculate on the same pixels following [7]. To solve the scale ambiguity, if the scale differed, the prediction was scaled with the median of the predicted data divided by the test data.

6.2 Data augmentation

Data augmentation is a well-used technique to both increase the dataset and improve the quality of the data. Using images in deep learning a lot of information is not only per pixel information but also its relation towards its neighbour and globally. For example, an image with an opposite moving car on the highway, the information about the car’s colour is not inherently important to the encoded feature space that is decoded to the depth. Same thing if the traffic is left-hand side traffic or right-hand side, the relative depth should not depend on it. Therefore, data augmentation can help with improving the quality of the data and helping the model not to be over reliant on a specific scenario. Most common data augmentation for images are for example flipping, rotating, scaling, and cropping. The augmentation is only applied on the training data and the augmentation that was used was brightness adjustment, saturation, contrast, and hue jitters. They were applied with 50 % chance and had a range of ± 0.2 units. The other augmentation was only applied on the input of the net and not when calculating the photometric error, since they would add unnecessary noise.

6.3 Implementations details

All models were trained on a single NVIDIA GeForce RTX 2080 GPU, with different batch size dependent on how much information was able to be fitted into memory. The Monodepth2, and models that is built on that core, was implemented with the PyTorch library, where the original code was released by its author. Some essential parts of MotionNet were implemented on top of the Monodepth2 code in PyTorch but was instead left in its original Tensorflow1 library and had to be changed to fit on a CUDA machine. All models were trained with the ADAM optimisation, $\beta_1 = 0.0, \beta_2 = 0.2$ and decreasing learning rate starting at 10^{-4} and reduced to 10^{-5} after half the training time. Monodepth2 and Manydepth were trained for 20 epochs and took about 12-20 hours to train, while MotionNet was trained for 15 epochs and took about 3 days. Monodepth2 and Manydepth both converged for all training sessions and produced a good depth after 1 epoch but the MotionNet did not always converge and was unstable. The nets were trained with resolution 192×640 , except for the MotionNet which was trained on a 128×416 resolution.

7 Results

Quantitative results of the different models are shown in Table 1-3 where the metric mention in the theory section is used. They are tested on the test dataset, labelled *All* for the whole set and the split datasets labelled *Static, Dynamic*. Where the split datasets has been annotated manually from the test dataset *All*. The results are compared with the baselines MD2 w. and w/o auto mask. The best and second-best result is shown in bold font. The metrics in blue represent that a lower result is better, while the orange represent the metrics where a higher result is better. The evaluation uses ground truth data and follows the procedure that [7] and other papers do, where the prediction is resized into the LIDAR resolution 352×1216 . The max depth was capped to 80 meter and the depth was scaled by the median of ground truth divided by the median of the predicted depth. This was to resolve the scale ambiguity and get the scale from the predicted truth to the ground truth from the LIDAR. The predicted pixels where only applied on what the LIDAR can see, since the LIDAR was on top of the station wagon, so a predefined crop was applied dependent on the spatial relation between the camera and the LIDAR.

Method - <i>All</i>	Time	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
MD2 w/o automask	t	0.156	2.192	5.078	0.220	0.840	0.950	0.974
MD2 w automask	t	0.096	0.934	4.107	0.167	0.919	0.968	0.983
MD2 softplus	t	0.095	0.888	4.081	0.165	0.919	0.969	0.984
MD2 flowmask	t	0.096	0.959	4.078	0.164	0.919	0.969	0.984
MD2 huber	t	0.094	0.850	4.104	0.165	0.917	0.968	0.984
MD2 uncertainty	t	0.096	0.979	4.210	0.166	0.919	0.968	0.983
MotionNet	t	0.112	1.654	4.682	0.183	0.901	0.959	0.978
Manydepth	t-1,t	0.090	0.897	4.022	0.159	0.921	0.970	0.984

Table 1: Monocular depth estimation the test dataset *All*. The best and second-best result is highlighted in bold. Manydepth makes prediction based on two frames, while the other uses only a single frame. The resolution for all predictions is 192×640 , except for MotionNet which had a resolution of 128×416 pixels. MD2 w/w.o automask is the baseline trained on the code given by its author.

Method - <i>Static</i>	Time	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
MD2 w/o automask	t	0.088	0.743	3.590	0.153	0.929	0.974	0.987
MD2 w automask	t	0.089	0.752	3.572	0.155	0.930	0.974	0.987
MD2 softplus	t	0.087	0.637	3.525	0.155	0.929	0.974	0.987
MD2 flowmask	t	0.087	0.715	3.532	0.152	0.929	0.975	0.988
MD2 huber	t	0.087	0.634	3.536	0.154	0.926	0.975	0.988
MD2 uncertainty	t	0.089	0.716	3.524	0.153	0.930	0.975	0.988
MotionNet	t	0.089	0.789	3.753	0.157	0.921	0.972	0.987
Manydepth	t-1,t	0.081	0.696	3.477	0.147	0.932	0.975	0.988

Table 2: Monocular depth estimation the test dataset *Static*. The best and second-best result is highlighted in bold. Manydepth makes prediction based on two frames, while the other uses only a single frame. The resolution for all predictions is 192×640 , except for MotionNet which had a resolution of 128×416 pixels. MD2 w/w.o automask is the baseline trained on the code given by its author.

Method - <i>Dynamic</i>	Time	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
MD2 w/o automask	t	0.203	7.164	7.130	0.245	0.864	0.929	0.953
MD2 w automask	t	0.124	1.996	5.671	0.204	0.886	0.955	0.975
MD2 softplus	t	0.126	1.944	5.621	0.204	0.886	0.955	0.975
MD2 flowmask	t	0.124	2.006	5.509	0.199	0.888	0.958	0.976
MD2 huber	t	0.121	1.861	5.564	0.200	0.891	0.956	0.976
MD2 uncertainty	t	0.124	2.082	5.855	0.204	0.897	0.954	0.973
MotionNet	t	0.205	6.013	7.407	0.270	0.845	0.919	0.946
Manydepth	t-1,t	0.112	1.766	5.392	0.189	0.905	0.961	0.978

Table 3: Monocular depth estimation the test dataset *Dynamic*. The best and second-best result is highlighted in bold. Manydepth makes prediction based on two frames, while the other uses only a single frame. The resolution for all predictions is 192×640 , except for MotionNet which had a resolution of 128×416 pixels. MD2 w/w.o automask is the baseline trained on the code given by its author.

The ablation study of the auto mask was as expected with that the metric score for the dynamic scene is worse without it than with it. It is also shown that MD2 with auto mask produces similar score for static scene as without. Monodepth2 [7] mentions that the auto mask shows improvement but shows it on a different dataset that is Eigen data split with the "no camera movement" frames (that is usually removed). The net however never shows it directly on explicit dynamic scenes. The softplus representation shows similar result for the dynamic scenes but show improvement on all metrics for the whole dataset. Since the net does not use hyperparameters for the disparity, the net can thus learn the representation by itself. We can see the

different representation in Figure 13 for a static scene where the disparity map is more sharp as one can see for example in the left trees. Otherwise for the different model the estimation is similar for the static scene. However, for the Manydepth the estimation for the back window of the left car is better estimated, since window can cause a mismatch from its transparent surface.

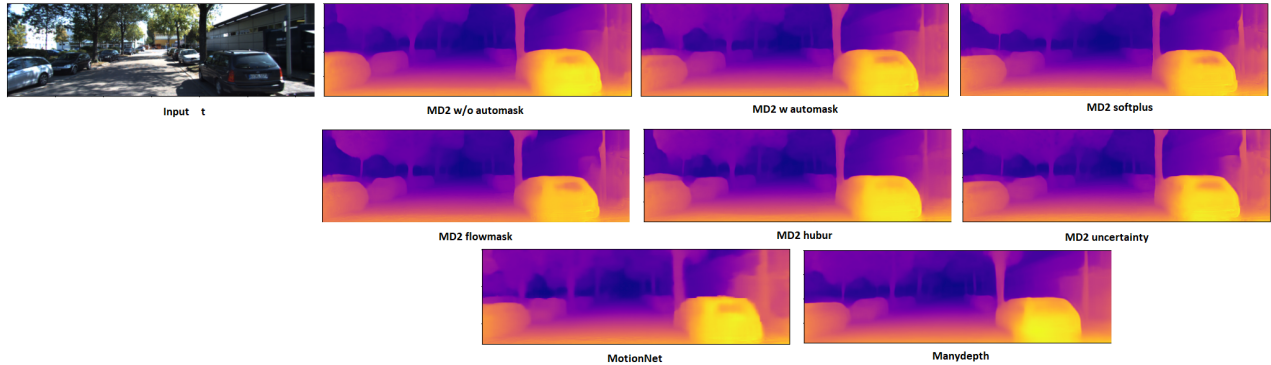


Figure 13: The disparity map for the different models, where the input is shown in colour and the disparity map is in a heatmap with lighter/more yellow for large disparity (small depth).

The result of the flow mask method for the dynamic dataset is better than the baseline, MD2 w automask, for 5 of the 7 metrics and are the best for 4 (excluding Manydepth). The mask is calculated by checking the backward-forward flow residual in HSV colour space and check where the residual is 80 % bigger than the mean of the residual. The flow depends on the two different independent depth estimation and the pose estimation. For dynamic scene the total pose estimation is dependent on a static scene and the appearance-based method for estimating depth also is worse where the motion happens. In Figure 14 the mask for the moving bicycle is masked out and the mask is black (removed from training) for which pixels in frame t is not eligible for training. The mask also includes the occluded part and the part where the depth is hard to estimate i.e. small sharp edges.

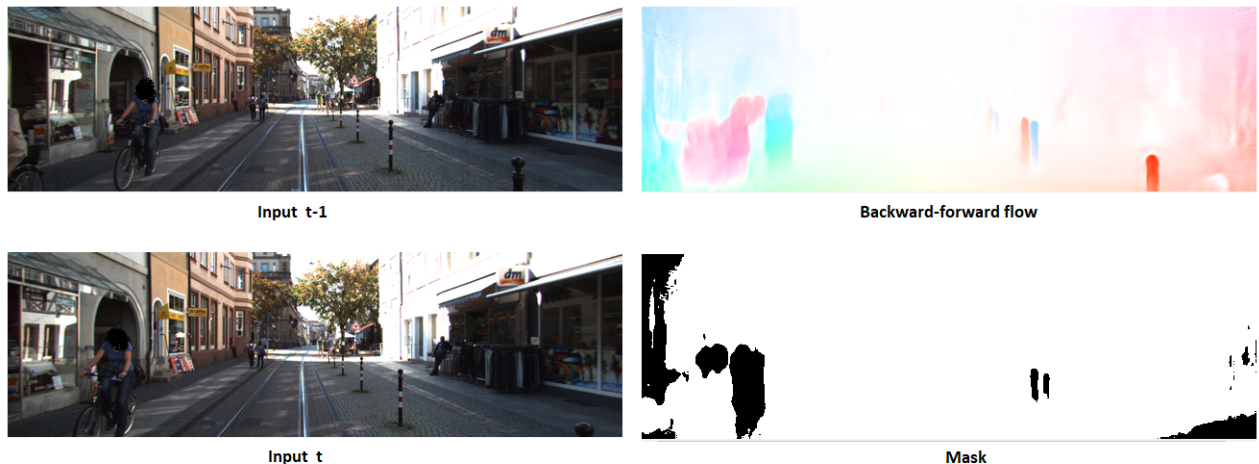


Figure 14: The flowmask for a dynamic scene, the backward-forward flow residual is shown in the top right and the resulting mask in the bottom right.

We can see in Figure 15 the mask also include part where the appearance matching is hard i.e. occlusion and repeated texture in the bush. It has a hard time finding the moving object (turning car) since the motion happens far away from the camera and is small.

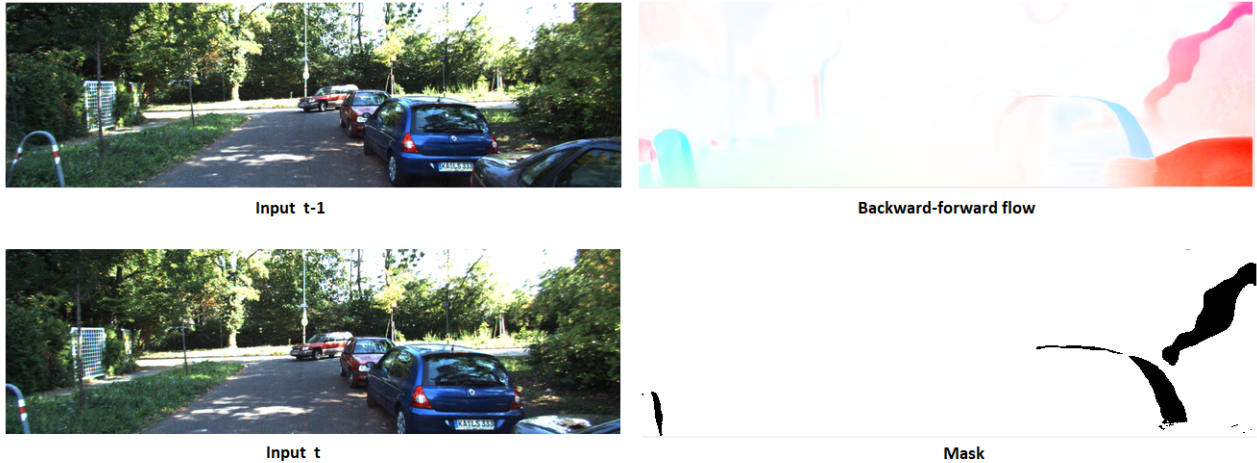


Figure 15: The flowmask for a static scene, the backward-forward flow residual is shown in the top right and the resulting mask in the bottom right.

Instead of the masking, the aleatoric uncertainty try to scale pixels where the depth is harder to estimate. In Figure 16 for the static scene the edges are hard to estimate correctly and for the dynamic the tree texture create uncertainty. The uncertainty can also find some uncertainty in the moving object behind the left car.

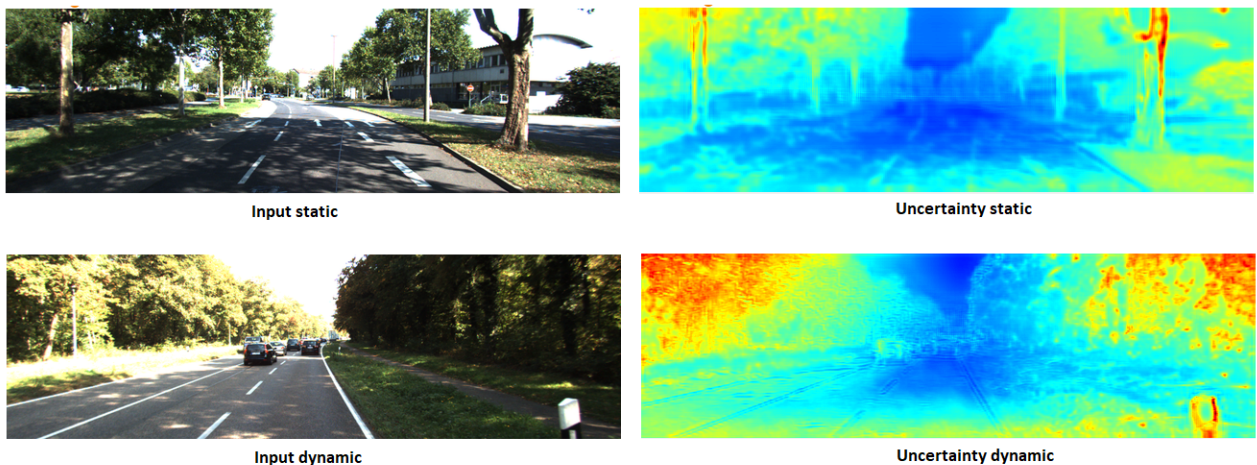


Figure 16: Aleatoric uncertainty for two different scenes.

8 Discussion

In this thesis a study is conducted on how and why the self-supervised depth estimation framework fails for moving objects. As shown, the baseline framework fails horribly when objects with respect to the camera movement is stationary and the depth is estimated as infinite. It also fails (during training) the appearance-based approach since a correct depth is not a correct match for moving objects and the weight is updated erroneously. The best approach found for the infinite depth was to mask out when the error from the same pixel for two frames is lower than the error from the reconstructed one i.e. the pixel appears stationary. Several monodepth frameworks does this approach, except for the MotionNet. They instead, in the appearance matching moves the stationary pixels that is caused from moving objects with the same velocity as the ego-motion. Other papers also separate out the worst frames (scenes in highway or stationary video stream) and does not train on them which significantly improves the infinite holes estimations. This works since

dynamic scenes mainly hinder the training step and not the inference step i.e. for most part, the moving objects is noise and does not hamper the generalisation of the net.

The approach to handle the mismatching from moving objects was either to estimate the motion and in the reconstruction, match the pixels so a correct depth correspond to a good match, or to mask out problematic pixels. The residual object translation with the auto masking would likely have the best outcome. However, in this work it was not possible to recreate the self-supervised method to estimate the per-pixel translation. From the original paper, the net only improves two of the seven metrics compared to the baseline Monodepth2 and that was tested on a custom resolution.

The other approach to handle object dynamics that showed result was to improve the data by masking out the bad matching either by a flow consistency or a scaling as huber or uncertainty. The flow consistency could be done differently since the backbone information is the depth estimation between the two frames and how they differ. An approach that the paper [2] do, is similar to the flow consistency mask and it is called the geometry consistency loss. They take two different depth predictions and take the difference as a loss to make it consistent and furthermore scale the photometric error with the difference. This requires a symmetric approach, which the baseline Monodepth2 currently does not support since the reconstruction always is source to target $t- > t'$. A way to add the symmetric approach would be to change the order from source to target to target to source, while keeping the sampling method and only inverse/transpose the pose matrix (since the pose matrix should be orthogonal). The reverse order switch is needed to keep the inverse sampling method the same, since there exists hardly any differentiable way to render the RGB-D image with a forward mapping (splatting instead of interpolation is needed). A way to incorporate a forward mapping could yield a better score. This could help the dynamic scene problem since forward mapping keeps geometry but creates unwanted holes, which the inverse mapping does not. Recently it has been proposed an approach to do forward mapping to increase performance for dynamic scenes. Since forward mapping holds the geometry, if we can do inverse mapping for all the non-moving objects and do forward mapping for the part of the image which has the moving objects, we could get a much better reconstructive image. The paper from KAIST [18] does this by segmenting out the moving objects and use the remained background image to predict the pose and with that prediction it predicts a depth map for the scene without any moving objects. It predicts the depth and motion for each segmented object and reconstruct the image with a nearest neighbour approach and minimises the hole that appears from the forward mapping. This solves the problem that moving objects influence the PoseNet and the mismatching in the reconstruction error. As the inverse mapping causes the pixel discrepancy on the moving object to be interpolated multiple times, causing a non-geometrical consistency in the objects and a worse reconstructed image.

The papers that handle the problem with monocular depth estimation are published up to the minute (less than 1 year before starting this thesis) and many handle the dynamic scene problem by explicitly computing optical flow, using segmented images or jointly learning another task related to depth. Since a goal of this thesis was to hold it close to the grander scheme and working within the Monodepth2 inference framework i.e. able to do prediction from a single image or from a shared encoded space, the possibility of using additional information was not studied. The use of different segmented or optical flow only in training could be used, but for the Monodepth2 baseline the paper [7] found no metric improvement for two of such proposals. The paper [20] try to lower the errors from dynamic scenes and improve the temporal depth consistency for objects. It uses optical flow to punish the spatial error, the disparity error from the optical flow and error from the warping. It estimates the optical flow by first running a homography-warp with an additional RANSAC to remove the ego-motion and then it uses a pretrained optical flow net on the reconstructed image and original image to find the residual flow. The results are slightly improved, when removing the scene with extreme dynamic motion and failed camera pose estimation, but for the whole dataset it did not achieve a better score then Monodepth2.

Manydepth has the best score for all the different metrics but are unfairly competing with net that only do inference from a single frame as it takes advantage of the temporal information. Also they require the PoseNet to run during inference as it adds the cost volume structure inside the encoder. This could break the grander scheme, which goes against the general purpose of this thesis. However there exist an important

information in the temporal information for the inference step that could be stored as an intermediate step. By saving to memory the latest temporal encoded feature it could in the decoder part be added as additional temporal information.

A natural problem with monocular camera is that it is not scale aware. One can easily solve this by saving, depending on the training data, the scale on which it was trained on. However, the scale that the different representation (softplus vs disparity) predicted was somewhat constant for different training and was roughly 12 for the softplus models and 4 for the disparity one. This difference could be the reason for the instability and the reason for non-multi-scaling for the softplus. A way to increase stability and remove scale ambiguity could be to use soft supervision from the LIDAR or regulate the pose. Toyota Research Institute (TRI) [9] has investigated how a sparse supervised signal, as low as 4 beams (roughly 100 data points in a 192×640 image), can both resolve the scale ambiguity and improve the result. They also use another backbone for their DepthNet called PackNet, which resolves the damaging downsampling that causes the dense final prediction to lose some local spatial information. This follows many other papers that try to update the old (2015) ResNet structure to other newer nets that do similar functions. There exists some downfall on the new nets as being too specific or having too long inference time, but PackNet is structurally close to ResNet and could thus help with the overall predictions. The net does not use down-/ upsampling pooling but have packing and unpacking block instead. So instead of passing the information lower in the spatial dimension by for example a maxpooling it packs it in the depth dimension instead. Things like this can help the overall framework.

MotionNet try to solve the dynamic scene reconstruction error by estimating all residual translation that cannot be described by the global translation. This is more aligned to how scene flow works. Scene flow net usually use flow cost volume additionally to the feature pyramid that FlowNet uses to combat the extra dimension. Since scene flow has a problem with separating depth from object translation. This is because given a depth, many scene flow translations along the parameterised line is projected to the same point. This causes the estimation of both depth and residual scene flow to be very hard and is probably the cause for why MotionNet collapses or is very sensitive. The approach with the $\frac{1}{2}$ -norm could solve the separation of depth and scene flow since it forces the object translation to be on a local scale with its spatial mean and thus limit the amount of reprojection along the line. The reason for convergence and its relation to the norm should be studied further. Another approach similar would be to assume good depth estimation, which often was the case after 1-2 epochs, and then train the residual translation between those depth estimations after warping. The input of the scene flow net could then be the two RGB-D images like MotionNet and have the global pose in the encoded feature space and the residual scene flow at the decoded space. Another possibility is to have the PoseNet and have the warped RGB-D image instead as an input. The problem with the optical/scene flow estimation is to remove the ego motion without influencing the depth estimation. It is hard to decouple the pose from the depth performance and vice versa.

As for ethical consideration, no personal information was used or stored. If there existed people on any shown images, the face was blurred.

9 Conclusion

In this thesis, the effect of dynamic scenes to the training of self-supervised monocular depth estimation was studied. By removing part of the image which appeared stationary to the camera, the result improved for all metrics. This was shown on annotate dataset, that had only images of varies dynamic scenes. The explicit check for improvement of the auto mask for dynamic scene has, from the best knowledge of the author, not been used before. The auto mask only handled cases of motion where the object travel in the same direction as the ego-motion. For arbitrary motion, the forward-backward consistency mask improved on six out of the seven metrics when compared to the Monodepth2 baseline. Additionally, a new depth representation was found out to work on the baseline and proved similar if not better results.

References

- [1] Jonathan T. Barron. *A General and Adaptive Robust Loss Function*. 2019.
- [2] Jia-Wang Bian, Huangying Zhan, Naiyan Wang, Zhichao Li, Le Zhang, Chunhua Shen, Ming-Ming Cheng and Ian Reid. *Unsupervised Scale-consistent Depth Learning from Video* 2021.
- [3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei. *Imagenet: A large-scale hierarchical image database* 2009.
- [4] Zhicheng Fang, Xiaoran Chen, Yuhua Chen and Luc Van Gool. *Towards Good Practice for CNN-Based Monocular Depth Estimation* 2020.
- [5] Andreas Geiger 2021. The KITTI Vision Benchmark Suite, accessed 28 may 2021, <<http://www.cvlibs.net/datasets/kitti/setup.php>>.
- [6] Andreas Geiger, Philip Lenz, Christoph Stiller and Raquel Urtasun. *Vision meets Robotics: The KITTI Dataset* 2013.
- [7] Clément Godard, Oisín Mac Aodha, Michael Firman and Gabriel Brostow. *Digging Into Self-Supervised Monocular Depth Estimation*. 2019.
- [8] Ariel Gordon, Hanhan Li, Rico Jonschkowski and Anelia Angelova. *Depth from Videos in the Wild: Unsupervised Monocular Depth Learning from Unknown Cameras* 2019.
- [9] Vitor Guizilini, Jie Li, Rares Ambrus, Sudeep Pillai and Adrien Gaidon. *Robust Semi-Supervised Monocular Depth Estimation with Reprojected Distances* 2019.
- [10] Antonio Gulli and Sujit Pal. *Deep Learning with Keras*. Packt Publishing, 2017.
- [11] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015.
- [12] Suzana Herculano-Houzel. *The human brain in numbers: a linearly scaled-up primate brain*. *Frontiers in Human Neuroscience* vol. 3, page 31, 2009.
- [13] Max Jaderberg, Karen Simonyan, Andrew Zisserman and Koray Kavukcuoglu. *Spatial Transformer Networks*. 2016.
- [14] Alex Kendall and Yarin Gal. *What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?* 2017.
- [15] Ue-Hwan Kim and Jong-Hwan Kim. *Revisiting Self-Supervised Monocular Depth Estimation*. 2021.
- [16] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017.
- [17] Maria Klodt and Andrea Vedaldi. *Supervising the new with the old: learning SFM from SFM* 2018.
- [18] Seokju Lee, Sunghoon Im, Stephen Lin and In So Kweon. *Learning Monocular Depth in Dynamic Scenes via Instance-Aware Projection Consistency* 2021.
- [19] Hanhan Li, Ariel Gordon, Hang Zhao, Vincent Casser and Anelia Angelova. *Unsupervised Monocular Depth Learning in Dynamic Scenes*. 2020.
- [20] Xuan Luo, Jia-Bin Huang, Richard Szeliski, Kevin Matzen and Johannes Kopf. *Consistent Video Depth Estimation* 2020.
- [21] Simon Meister, Junhwa Hur and Stefan Roth. *UnFlow: Unsupervised Learning of Optical Flow with a Bidirectional Census Loss* 2017.
- [22] Chaoyang Wang, Jose Miguel Buenaposada, Rui Zhu and Simon Lucey. *Learning Depth from Monocular Videos using Direct Methods*. 2017.

- [23] Jamie Watson, Oisin Mac Aodha, Victor Prisacariu, Gabriel Brostow and Michael Firman. *The Temporal Opportunist: Self-Supervised Multi-Frame Monocular Depth* 2021.
- [24] Qizhe Xie, Minh-Thang Luong, Eduard Hovy and Quoc V. Le. *Self-training with Noisy Student improves ImageNet classification*. 2020.
- [25] Tinghui Zhou, Matthew Brown, Noah Snavely and David G. Lowe. *Unsupervised Learning of Depth and Ego-Motion from Video*. 2017.