Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Anders Thallaug Fagerli

Deep Monocular Depth Estimation for Autonomous Underwater Vehicles

Master's thesis in Cybernetics and Robotics
Supervisor: Annette Stahl
Co-supervisor: Mauhing Yip, Rudolf Mester
June 2022

NTNU
Norwegian University of Science and Technology

aros vision group

Anders Thallaug Fagerli

Deep Monocular Depth Estimation for Autonomous Underwater Vehicles

Master's thesis in Cybernetics and Robotics
Supervisor: Annette Stahl
Co-supervisor: Mauhing Yip, Rudolf Mester
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

Exploring and utilizing the world's oceans in a sustainable manner is crucial for solving challenges such as global warming. Autonomous underwater robots aim to replace human operators in dangerous and harsh underwater environments while efficiently and sustainably surveying, exploring and acting on underwater structures. Essential for autonomy is the ability to map the surroundings, and this thesis explores the mapping of underwater environments with a monocular camera using deep learning.

A conditional variational autoencoder (CVAE) conditioned on the deep features of images is implemented as a baseline for estimating the depth of scenes, trained in a supervised fashion on synthetic underwater images. The aleatoric uncertainty of its predictions is additionally estimated, allowing the predictions to be continuously evaluated and fused with other probabilistic models. Several modifications to the baseline architecture are proposed to improve depth estimation in underwater environments. An edge detector based on the estimated aleatoric uncertainty is derived, allowing smoothness priors on otherwise unsuited data for geometric smoothing. A novel method for incorporating auxiliary sparse depth is proposed, fusing sparse data at multiple scales in a late fusion scheme for convolutional neural networks. Spatiotemporal networks are also investigated, using multiple temporally adjacent images as input to the model. A pretraining scheme for learning motion filters in 2D spatiotemporal networks is presented, and the generation of geometrically consistent depth estimates between multiple views is also explored. Results on a synthetic, photorealistic underwater dataset show that many of the proposed modifications improve the performance of the baseline, both in terms of qualitative and quantitative results. However, the model is only evaluated on synthetic data, and it remains to be seen how the proposed modifications affect performance in actual underwater environments.

Sammendrag

Å utforske og utnytte verdenshavene på en bærekraftig måte er avgjørende for å løse utfordringer som global oppvarming. Autonome undervannsroboter har som hensikt å erstatte menneskelige operatører i farlige undervannsmiljøer, samtidig som de effektivt kartlegger, utforsker og manipulerer undervannsstrukturer. Avgjørende for autonomi er evnen til å kartlegge omgivelsene, og denne oppgaven utforsker kartlegging av undervannsmiljøer med et monokulært kamera ved hjelp av dyp læring.

En variasjonsautokoder betinget på bilder er implementert som en grunnlinje for å estimere dybden til scener, trent på syntetiske undervannsbilder med sann dybdedata. Den aleatoriske usikkerheten til prediksjonene blir i tillegg estimert, slik at prediksjonene kontinuerlig kan evalueres og brukes sammen med andre sannsynlighetsmodeller. Flere modifikasjoner av grunnlinjearkitekturen er foreslått, med den hensikt å forbedre dybdeestimering i undervannsmiljøer. En kantdetektor basert på den estimerte aleatoriske usikkerheten utledes, og tillater metoder for geometrisk utjevning for data som ellers er uegnet. I tillegg foreslås en ny metode for å ta i bruk sparsomlige målte dybdeestimater som ekstra data, der dataen slås sammen på flere skalaer i konvolusjonelle nevrale nettverk. Spatiotemporale nettverk blir også undersøkt, ved å ta i bruk flere temporale sammenhengende bilder som input til modellen. En metode for å lære bevegelsesfiltre i 2D spatiotemporale nettverk presenteres, og generering av geometrisk konsistente dybdeestimater mellom flere bilder blir også utforsket. Resultater på et syntetisk, fotorealistisk undervannsdatasett viser at mange av de foreslåtte modifikasjonene forbedrer ytelsen til grunnlinjen, både når det gjelder kvalitative og kvantitative resultater. Modellen er imidlertid kun evaluert på syntetiske data, og det gjenstår å se hvordan de foreslåtte modifikasjonene påvirker ytelsen i virkelige undervannsmiljøer.

Preface

This Master's Thesis concludes a Master of Science for the study program Cybernetics and Robotics at the Norwegian University of Science and Technology, counting for 30 credit points. The work is conducted for the Autonomous Robots for Ocean Sustainability project, supported by the Research Council of Norway (project number: 304667). This thesis is a continuation of TTK4550, and will have elements that are highly similar to the specialization project report [1].

This marks the end of five years at Cybernetics and Robotics and seven years in Trondheim. Although it has been extremely challenging at times, it has also been some of my best and most rewarding years. I couldn't have gotten through them without friends and family, so a huge thanks go to them. Working on this thesis has been especially challenging, so I would like to thank my supervisors, Annette Stahl, Mauhing Yip and Rudolf Mester, and also the rest of the AROS Vision Group for excellent feedback and discussions throughout the year.

Anders Thallaug Fagerli
Trondheim, June 6, 2022

Table of Contents

List of Tables

List of Figures

List of Abbreviations

APS Average performance score
ARD Absolute relative difference
AROS Autonomous Robots for Ocean Sustainability
AUV Autonomous underwater vehicle

CNN Convolutional neural network
CVAE Conditional variational autoencoder

KL Kullback-Leibler

LOGRMSE Root mean squared log error

MAP Maximum a posteriori

RMSE Root mean squared error

SFM Structure from motion
SGVB Stochastic gradient variational bayes
SLAM Simultaneous localization and mapping
SRD Squared relative difference

VAE Variational autoencoder
VSLAM Visual simultaneous localization and mapping

1 | Introduction

Autonomous robots have in recent years demonstrated increasingly complex capabilities for a variety of tasks, both previously and currently performed by human operators. Aiming to close the gap between humans and robots, the robotics community incrementally makes technological advancements towards a future where robots replace humans in everyday tasks, but importantly also in dangerous and harsh environments with a high risk to human lives. The Autonomous Robots for Ocean Sustainability project aims to replace human operators in underwater operations, automating tasks such as underwater visual inspection, detection and surveying of pipelines, fish farms and oil and gas facilities. The visual perception of a robot plays a vital role for its working condition in whatever environment it is placed in, much like a human's, and is commonly used to both map the surroundings and localize the robot in its surroundings, the two being highly interconnected. Mapping the environment is therefore an essential part of any robot's perceptual system. In this work, the focus is on developing full (dense) maps of underwater scenes, reconstructing the complete geometry by sensory information, in this case, from a monocular camera.

1.1 Motivation

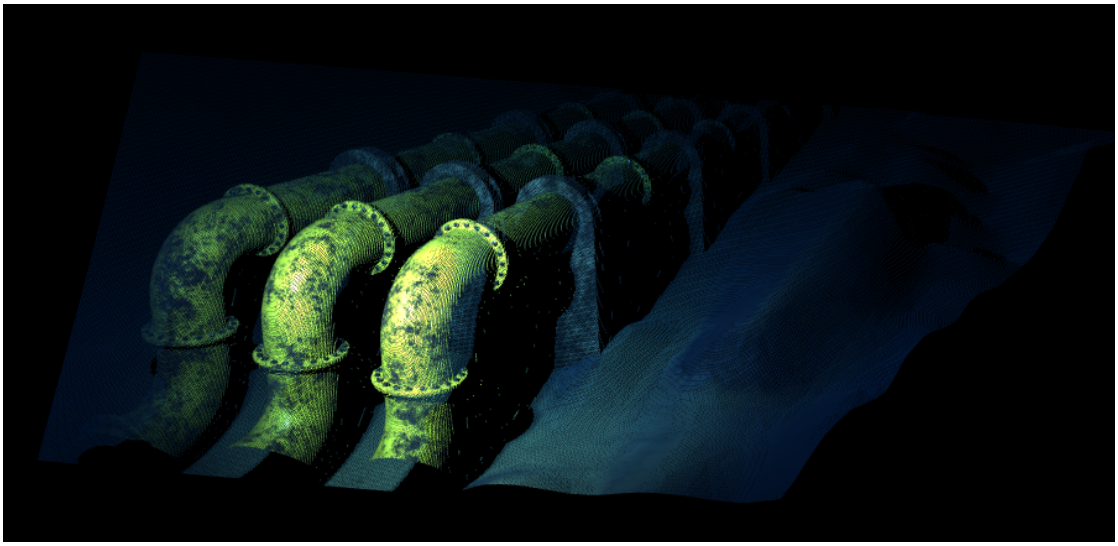
In 2019, the Norwegian University of Science and Technology launched a research project on sustainable exploration and utilization of the world's oceans called Autonomous Robots for Ocean Sustainability (AROS). This project aims to address and solve challenges relating oceans and how we use them to issues such as global warming, in addition to sustainable energy consumption and harvesting of resources like food and minerals for an ever-increasing global population. As the project name suggests, these sustainability goals will be achieved by employing *autonomous robots* to replace humans or remotely operated vehicles in underwater operations. Central to the fulfillment of AROS is, therefore, the design and implementation of autonomous underwater vehicles (AUVs) with perceptual, navigational and manipulable capabilities that satisfy the constraints of the project.

Among these capabilities, the perceptual system of an AUV is perhaps most affected by the harsh and difficult underwater environment. Perceiving sensors like LiDAR, radar and many other electromagnetic sensors fail due to the sensing signals being absorbed in water, and accurate sonars pose threats to marine life due to the high-frequency sound waves. Lack of visible natural light makes a challenge for cameras, requiring artificial light to provide sufficient illumination of the scenes. An example is seen in Figure 1.1, where an AUV uses headlights for inspecting the underwater environment. However, the strength of cameras is that both contextual and geometric information can be retrieved by the images they capture. In Figure 1.2, an

Figure 1.1: *Inspection of underwater structures using the Eelume robot. Image courtesy:* `eelume.com`.

(a) *Underwater RGB image.*

(b) *Ground truth depth map.*



(c) *3D geometry represented as a point cloud.*

Figure 1.2: *Synthetic underwater structure visualized by three different representations.*

underwater image is captured by illuminating the scene, and the 3D geometry of the scene can subsequently be estimated from the image. This provides a measure of proximity, namely the *depth* of the scene, in addition to cues and semantics from the image itself. Combined, as shown in Figure 1.2c, the result is a more information-rich structure than other sensors can capture. Note that this structure is generated from artificial data and not estimated from the image.

The generation of the 3D structure of a robot's surroundings is referred to as *mapping* and is crucial for a robot to act correctly in its environment. It is thus important for external tasks but also to enable autonomy itself. In visual simultaneous localization and mapping (VSLAM), the robot cannot localize itself without concurrently mapping. Subsequently, under guidance, navigation and control, the robot's location must be known, and thereby also the map. For the robot to move safely in its environment, it must also detect and avoid obstacles in its path. These are just some examples of why mapping is necessary for autonomy, not to mention external functions such as surveying, inspection and intervention of underwater structures.

The greatest challenge with using cameras is *estimating* the geometry from the images they take. In contrast to other proximity sensors, the depth of the scenes is not measured directly and must be reconstructed using some computational algorithm, usually utilizing multiple views. It is common to classify structure estimation problems as either *sparse* or *dense*, where sparse methods only estimate a subset of points in the scene and dense methods estimate all points in the scene. For the sole purpose of mapping, it is clear that a dense reconstruction is desired, as a sparse geometry will not give sufficient information for tasks such as visual inspection and collision avoidance. A challenge with dense methods, however, is that they are usually *direct*, meaning they operate on image intensity values directly. An assumption for most of these methods is photometric consistency between views, namely that the intensity value of a pixel viewing a point in one view should be the same as the corresponding pixel viewing the same point in another view. This is an approximation to the real world, which holds for many above-water scenarios, but fails underwater. Figure 1.2a shows one of the failure modes, where the artificial illumination from the robot gives a non-uniform scattering of light in the image. Traditional, model-based methods will therefore typically fail underwater without additional modifications to the algorithms.

With the recent development in deep learning, powerful new models capable of estimating dense depth from single views have emerged, motivating their use in the otherwise difficult underwater environment. The deployment of deep learning in this scenario has especially two motivating reasons:

- Deep learning models can estimate dense depth from single views, enabling monocular setups in AUVs with tight physical constraints that do not allow stereo configurations.
- Deep learning models may learn photometric relationships between photometrically inconsistent views, enabling multiple view methods.

This work will, therefore, investigate monocular dense depth estimation in underwater environments using deep learning.

1.2 Problem description

The objective of this work is rooted in the objective of AROS, and more specifically, investigating methods for densely mapping the environment of an AUV using monocular cameras. The current

state-of-the-art estimates the complete geometry by stereo vision, often relying on the absence of photometric and geometric distortions, but many robots operate in conditions that prohibit both stereo configurations and the assumption of no distortions. Methods based on deep learning have recently shown promise for above-water monocular depth estimation but have yet to be trusted in more visually harsh underwater environments. The following is an extension of the author's previous work [1], and the subtasks considered in this follow-up are, therefore:

1. Propose methods and modifications for improving depth estimation performance on data realistically relating to the targeted underwater environment.
2. Implement the proposed methods and modifications.
3. Test and verify the implementation on relevant datasets.

Considerations such as real-time performance and machine-specific implementation will not be investigated to limit the scope of the thesis. Additionally, backed by the conclusion of [1], actual underwater data will not be considered or reflected on in this work, as there is currently no underwater dataset that provides the necessary ground truth depth. Self-supervised methods may operate even in these cases, but the photometric and geometric distortions in underwater data render these methods futile. The scope of verification and testing is, therefore, limited to artificial environments, and the methods are limited to supervised methods.

1.3 Contributions

The contributions of this thesis are in summary:

- The *uncertainty-aware smoothness* as an improvement on the commonly used edge-aware smoothness.
- *Multi-scale sparse depth* for improving the integration of auxiliary sparse depth in depth estimation networks, compared to other established methods.
- *Motion filter learning* for improving the convergence and learning of spatiotemporal 2D convolutional neural networks.
- The *depth-conditioned variational autoencoder*, as an improvement on variational autoencoders conditioned on uncertainty for depth estimation.
- An evaluation of the VAROS Synthetic Underwater dataset for deep learning-based methods.

Additionally, the source code for all developed methods is made public at

`https://github.com/andersfagerli/Depth-CVAE`.

1.4 Thesis outline

Chapter 2 presents background theory and related work, defining some of the necessary notation and formulas for later use. In Chapter 3, the proposed methods and improvements for deep monocular depth estimation are presented, with results and discussion for all experiments displayed in Chapter 4. Chapter 5 concludes the thesis, with remarks on possible further work.

2 | Background

This chapter presents the necessary background theory and details for going forward in subsequent chapters. Preliminary theory on machine learning and deep learning is skipped for brevity, and the reader is assumed to have some knowledge of basic theory related to neural networks, and especially convolutional neural networks (CNN), so only specifics relevant for the chosen architecture are presented here. First, a brief description relating depth to cameras and the world is given, only providing the necessary details for later use. Then, a more detailed background theory on the implemented architecture is described. Methods to evaluate a deep learning model's uncertainty are then presented before some common performance metrics for depth estimation are given. Finally, the necessary background is rounded off with the related work in the literature.

2.1 Camera geometry

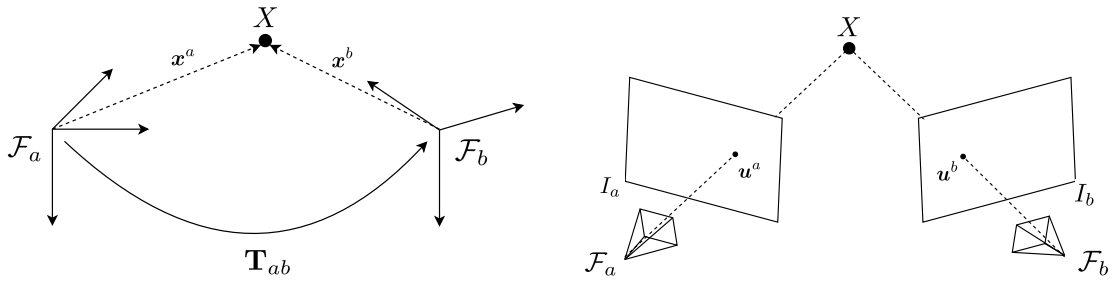
The following relates depth to the three-dimensional geometry of the world and how a camera might use it for capturing images of scenes and subsequently capture scenes from images. Most of the presented theory is based on [2], with notation borrowed from [3].

2.1.1 3D geometry

The world as we know it is a three-dimensional physical space, most commonly represented mathematically as Euclidean space, with each point located by its Cartesian coordinates. Denoting a point in the world by X, its coordinate vector $\boldsymbol{x} \in \mathbb{R}^3$ represents the point's displacement from some coordinate frame $\mathcal{F}$. As the location is *relative* to the frame, the frame and point must be specified by denoting that, e.g., $\boldsymbol{x}^a$ belongs to frame a, $\mathcal{F}_a$.

Another useful representation is the projective space, translating to the later-described camera model, where homogeneous coordinates describe the projection of 3D points on a plane. If a Cartesian coordinate vector $\boldsymbol{x} = [x \;\; y \;\; z]^\mathsf{T}$, then its homogeneous representation in projective space is $\tilde{\boldsymbol{x}} = \lambda[x \;\; y \;\; z \;\; 1] \in \mathbb{P}^3$, where λ is some scaling factor. Homogeneous coordinates inhabit useful properties for projective geometry and will see usage for simplifying transformations and camera projections.

When observing the world from multiple frames, it may be necessary to know how these frames are positioned and oriented relative to each other. It is, for example, common to have a fixed world frame for representing the world and a local coordinate frame representing a camera moving in this world. When the camera observes points in its local frame, it may be of interest to transform these into the world frame, requiring the position and orientation of the camera with



(a) *The point X expressed by $\boldsymbol{x}^a$ and $\boldsymbol{x}^b$ in $\mathcal{F}_a$ and $\mathcal{F}_b$, respectively, with the pose $\mathbf{T}_{ab}$ of $\mathcal{F}_b$ relative to $\mathcal{F}_a$.*

(b) *The point X being projected onto the image planes I_a and I_b at pixels $\boldsymbol{u}^a$ and $\boldsymbol{u}^b$, respectively.*

Figure 2.1: *The 3D geometry in Euclidean space (left) and the projection of 3D points into 2D image coordinates (right).*

respect to the world. The pose

$$\mathbf{T}_{ab} = \begin{bmatrix} \mathbf{R}_{ab} & \boldsymbol{t}_{ab}^a \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \in SE(3) \tag{2.1}$$

is a homogeneous transformation matrix that describes the rotation $\mathbf{R}_{ab} \in SO(3)$ and translation $\boldsymbol{t}_{ab}^a \in \mathbb{R}^3$ from $\mathcal{F}_b$ to $\mathcal{F}_a$, where $SE(3)$ and $SO(3)$ are the *special Euclidean* and *special orthogonal* Lie groups in 3D, respectively. The transformation in (2.1) can then be used to transform points from one frame to another, as

$$\tilde{\boldsymbol{x}}^a = \mathbf{T}_{ab}\tilde{\boldsymbol{x}}^b. \tag{2.2}$$

In Figure 2.1a, two frames and their relative pose is shown, which can be used to transform $\boldsymbol{x}^b$ into $\mathcal{F}_a$ by (2.2). Note that the coordinate vectors in (2.2) are homogeneous, but the mapping to and from Cartesian and homogeneous coordinates for our purposes is simply

$$\tilde{\boldsymbol{x}} = [x \quad y \quad z \quad 1]^\mathsf{T} \quad \mapsto \quad \boldsymbol{x} = [x \quad y \quad z]^\mathsf{T}, \tag{2.3a}$$

$$\boldsymbol{x} = [x \quad y \quad z]^\mathsf{T} \quad \mapsto \quad \tilde{\boldsymbol{x}} = [x \quad y \quad z \quad 1]^\mathsf{T}. \tag{2.3b}$$

2.1.2 The perspective camera model

The perspective camera model, also called the pinhole model, describes the relationship between 3D points in the world and their corresponding projection onto the image plane of a camera, and is one of several possible models for a camera. Generally speaking, a camera projection is a mapping $\pi : \mathbb{R}^3 \to \Omega$ that transforms 3D points in the camera frame to pixels $\boldsymbol{u} \in \Omega$ in the image plane,

$$\boldsymbol{u} = \begin{bmatrix} u \\ v \end{bmatrix} = \pi(\boldsymbol{x}). \tag{2.4}$$

Figure 2.1b shows two cameras viewing the same 3D point X, and the respective projections onto each image plane. Inversely, the *backprojection* is a mapping $\pi^{-1} : \Omega \times \mathbb{R}^+ \to \mathbb{R}^3$ that transforms pixels back to 3D points,

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \pi^{-1}(\boldsymbol{u}, z), \tag{2.5}$$

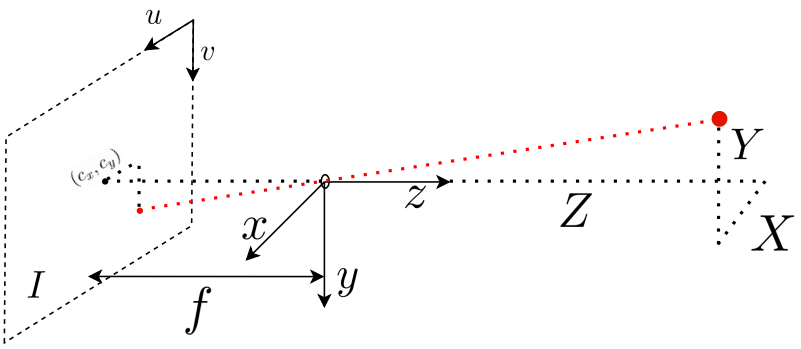


Figure 2.2: *The perspective camera, projecting a point $\boldsymbol{x} = [X\ Y\ Z]^\mathsf{T}$ through the projective center of the camera and onto the image plane I located at $z = -f$, where f is the focal length of the camera. The projective center has pixel coordinates $u = c_x$ and $v = c_y$ in the image plane.*

which also requires the depth, z. The exact forms of (2.4) and (2.5) for the perspective camera will be presented shortly.

Same as for the homogeneous representations of 3D geometry, a 2D Cartesian vector may be mapped from and to its homogeneous representation by

$$\tilde{\boldsymbol{u}} = [u \quad v \quad 1]^\mathsf{T} \quad \mapsto \quad \boldsymbol{u} = [u \quad v]^\mathsf{T}, \tag{2.6a}$$

$$\boldsymbol{u} = [u \quad v]^\mathsf{T} \quad \mapsto \quad \tilde{\boldsymbol{u}} = [u \quad v \quad 1]^\mathsf{T}, \tag{2.6b}$$

which allows some useful properties for describing camera projections.

The full perspective camera model can finally be described using the notation above, and can be seen more clearly in Figure 2.2. Denoting $f_x = f s_x$ and $f_y = f s_y$ to be the size of unit length in horizontal and vertical pixels, respectively, where f is the focal length in metric units and s_x and s_y are the pixel densities per metric unit, it follows by the similar triangles in Figure 2.2 that

$$\frac{u - c_x}{f_x} = \frac{X}{Z} \quad \rightarrow \quad u = f_x \frac{X}{Z} + c_x, \tag{2.7a}$$

$$\frac{v - c_y}{f_y} = \frac{Y}{Z} \quad \rightarrow \quad v = f_y \frac{Y}{Z} + c_y. \tag{2.7b}$$

The projections in (2.7) can more compactly be represented by using the camera intrinsic matrix,

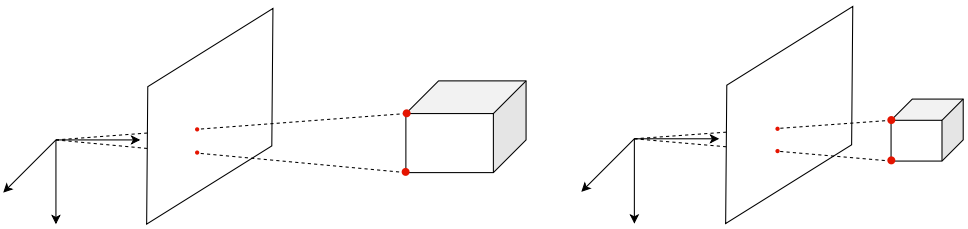
$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.8}$$

where there is assumed no skew between the axes in the image plane, such that the perspective camera projection function is

$$\boldsymbol{u} = \pi_p(\boldsymbol{x}; \mathbf{K}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{K} \frac{\boldsymbol{x}}{z}. \tag{2.9}$$

The perspective camera backprojection function can now be written using homogeneous pixel coordinates as

$$\boldsymbol{x} = \pi_p^{-1}(\boldsymbol{u}, z; \mathbf{K}) = z \mathbf{K}^{-1} \tilde{\boldsymbol{u}}. \tag{2.10}$$



(a) *The projection of a large cube on the image plane.* ***(b)*** *The projection of a small cube on the image plane.*

Figure 2.3: *Projections of two differently sized cubes intersect the image plane at the exact same location, due to the different distances from the camera. Figure is inspired by [2].*

Revisiting Figure 2.1b, it may be of interest to find corresponding pixels between two images that view the same point in space. The warp function

$$\boldsymbol{u}^a = w(\boldsymbol{u}^b, z^b, \mathbf{T}_{ab}) = \pi_p(\mathbf{T}_{ab} \cdot \pi_p^{-1}(\boldsymbol{u}^b, z^b)) \tag{2.11}$$

maps a pixel in I_b to its corresponding pixel in I_a using the known transformation $\mathbf{T}_{ab}$ between the two frames, the intrinsics of the cameras and the depth of the point.

2.1.3 From 2D to 3D

This work aims to infer the three-dimensional geometry of the world from two-dimensional images. The backprojection function (2.10), however, seems to solve this exact goal, as it takes a 2D pixel coordinate and maps it to a 3D point but does so using the points *depth*. In other words, knowledge of the scene's geometry must already be known before a pixel can be backprojected. Although knowledge may come in many forms, it is clear that the scene can be reconstructed once the depth is known. Going from 2D to 3D will therefore be analogous to finding the depth of the scene.

This problem is perhaps one of the most researched within computer vision and is a so-called *inverse problem*, that is, determining from pixels the causal points that produced them. Being inherently ill-posed, the problem arises because one entire dimension is lost when going from 3D to 2D, and today there is no model-based solution using one single image. Figure 2.3 illustrates one of the reasons why this problem perhaps is impossible to solve analytically, where two cubes at different distances from the camera have the same projection in the image plane. For every image, there is, in fact, an infinite number of plausible scenes, informally proven for the example above, where the cube can vary its size and distance to the camera accordingly in an infinite number of ways to produce the same image.

Today there are numerous approximations to the problem's solution, where most fall under the category of multiple view geometry. However, none truly solve it as they impose additional assumptions or numerical approximations. The details of multiple view geometry are skipped here, as they are not relevant for further discussion, but it should be noted that it is *the* established way of estimating the scene from images. A good starting point for more on multiple view geometry is [2].

Instead of going down the road of multiple view geometry, this work focuses on estimating depth from single views. Notice that the phrasing of the problem has turned to estimation rather than some analytical solution. As a hypothesis, let there be some (unknown) function $f : \mathbb{R} \to \mathbb{R}^3$

that maps a pixel value to a valid 3D point,

$$\boldsymbol{x} = f(I(\boldsymbol{u})), \tag{2.12}$$

here using monochrome images as an example. Mark the difference from (2.10) in that (2.12) is not dependent on depth, and solely maps a pixel *value* to a 3D point. The goal is to find some approximation, $g(I(\boldsymbol{u}))$, such that

$$\hat{\boldsymbol{x}} = g(I(\boldsymbol{u})) \approx f(I(\boldsymbol{u})). \tag{2.13}$$

Although more traditional model-based ways of finding the approximation may exist, recent years have seen that methods based on machine learning find this approximation to an adequate extent, where deep convolutional neural networks are typically used as $g(\cdot)$. Although limited to a restricted domain of scenes, not a one-size-fits-all solution, they learn mappings in incredibly harsh environments where traditional methods typically fail due to broken assumptions. Varying illumination, non-Lambertian surfaces and other photometric or geometric distortions are typical for scenes where methods based on multiple view geometry fail, but deep learning does not. In combination with the ability to satisfy (2.13) without any prior knowledge of the scene, this makes deep learning suitable for the problem at hand.

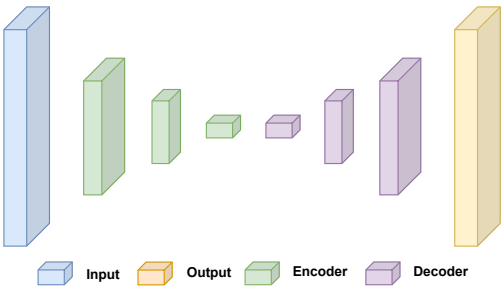
2.2 Autoencoders

Autoencoders have become a standard for fine-grained classification and regression tasks, such as semantic segmentation and depth estimation. They typically consist of a contracting path, called the *encoder*, and an expanding path, called the *decoder*. The encoder shares the structure of typical classification networks[4]–[6]. It is used to capture global context, while the decoder is used for enabling fine-grained predictions, often in the same resolution as the input. The problem is that local context, or localization of high-resolution features, is lost in the contracting path, giving coarse predictions. A common approach to solving this is to combine feature maps in the contracting path and the feature maps in the expanding path. This was popularly introduced in [7] for segmentation, but already in [8] an autoencoder-like CNN for depth estimation was used in the same manner. The most prevalent implementations using this structure for depth estimation are [8] and [9]. Most of this section is taken from the author's previous work, [1].

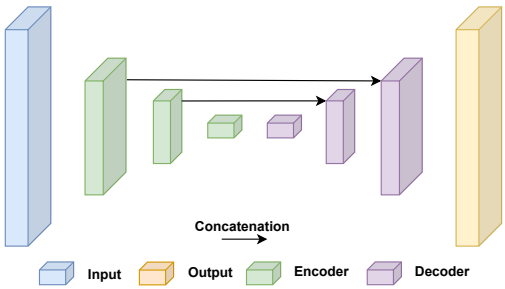
2.2.1 Vanilla autoencoders

A vanilla autoencoder is typically used to learn data encodings and not for inference directly. An example of an autoencoder implemented as a CNN is shown in Figure 2.4a. The encoding, typically called the *latent representation*, is a lower-dimensional feature that can be interpreted as a compressed version of the input. This is useful for reducing the number of features that describe the data, but this is in the domain of dimensionality reduction and not directly helpful in inferring depth from images. As it aims to encode the data, it is trained on reconstructing the input at the output, minimizing the *reconstruction loss*, giving an encoding that can be decoded into the original data. This is not the aim in this particular chapter and will therefore not be explained further here, but is revisited in Chapter 2.2.2 where it is of more relevance.

Autoencoders are commonly used for fine-grained predictions, either for classification or re-

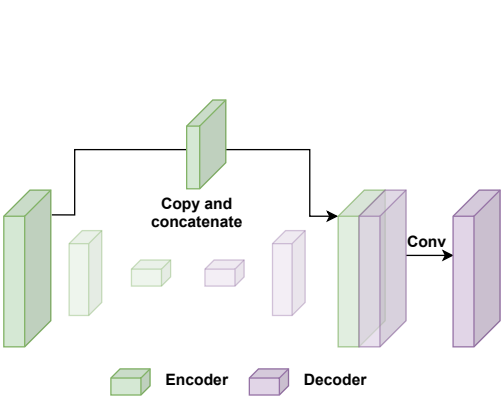


(a) Vanilla autoencoder composed of an encoder and a decoder.

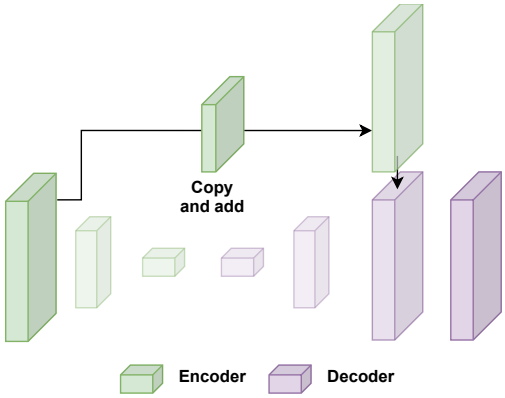


(b) Autoencoder with connections between the encoder and decoder for fine-grained predictions.

Figure 2.4: *Graphical representations of CNN-based autoencoders.*



(a) *Concatenation of feature maps.*



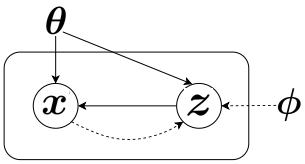
(b) *Addition of feature maps.*

Figure 2.5: *Different connection types between encoder and decoder.*

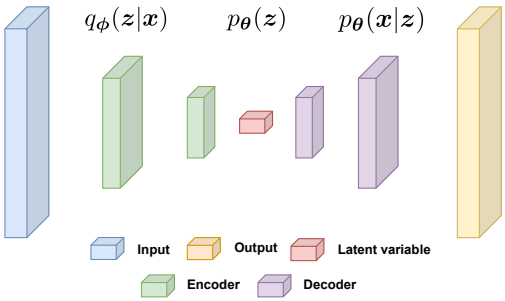
gression. This is partly possible due to the expanding path in the decoder, but also because high dimensional features from the encoder can be added to the decoder, as shown in Figure 2.4b. This is usually done by concatenating feature maps channel-wise in the decoder, as displayed in Figure 2.5a. This increases the number of channels in the respective decoder layer, and a subsequent convolution is usually performed to reduce the channels afterward, producing the final decoder feature map. An alternative is to instead add feature maps, as in Figure 2.5b. The connections effectively give the decoder more information about the features of the input, and by connecting higher-dimensional feature maps, more fine-grained information is retained in the decoder.

2.2.2 Variational autoencoders

First introduced by [10], and later adapted for depth estimation by [11], the variational autoencoder (VAE) adds a variational component to the previously described autoencoder architecture to enable generation of new data. The VAE is originally rooted in variational inference for probabilistic models and not within deep learning directly, but is often implemented by a neural network. The following chapters will therefore describe the VAE from a probabilistic perspective, but with emphasis on the practical implementation using neural networks. Finally, it is shown how a VAE can be used for estimating depth from monocular images, with the crucial properties it inhabits for further dense structure from motion (SfM) and VSLAM.



(a) Bayesian network of the model. Solid lines are the generative model, $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{z})$, while dashed lines are the variational approximation $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. Figure is inspired by [10].



(b) Model implemented as a CNN using the autoencoder architecture, with distributions corresponding to encoder, latent variable and decoder.

Figure 2.6: *Different graphical representations of the directed probabilistic model under consideration.*

Variational inference with intractable latent posteriors

The motivation of [10] is to solve inference problems that involve directed probabilistic models with latent variables whose posterior distribution is intractable. For observable variables $\boldsymbol{x}$ and latent (unobservable) variables $\boldsymbol{z}$, the described model will contain probability distributions $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$, $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, all parameterized by the model parameters $\boldsymbol{\theta}$. In the context of data generation, a new data point $\boldsymbol{x}$ is sampled from the distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, so in this case the objective is to find $p_{\boldsymbol{\theta}}(\boldsymbol{x})$. Using the law of total probability,

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z}) d\boldsymbol{z}, \tag{2.14}$$

the latent variable $\boldsymbol{z}$ can be used to infer knowledge of $\boldsymbol{x}$. The question is now how to find the correct latent distribution and mapping from $\boldsymbol{z}$ to $\boldsymbol{x}$, and how to solve the (generally intractable) integral in (2.14). The idea is to sample $\boldsymbol{z}$ that produce $\boldsymbol{x}$, using some distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ to give the $\boldsymbol{z}$ that are likely under $\boldsymbol{x}$. Using Bayes' rule,

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}, \tag{2.15}$$

it is seen that this distribution in the general case is intractable to solve, as it includes the intractable (2.14). Variational Bayesian methods solve this by the approximation $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \approx p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, parametrized by $\boldsymbol{\phi}$, where the aim is to find a distribution that is close to the intractable posterior. This is typically done by finding the parameters $\boldsymbol{\phi}$ that minimize the difference between $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. A graphical representation of the model can be seen in Figure 2.6a.

Relating the above to neural networks, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is denoted as the encoder and $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ as the decoder in an autoencoder, with $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ being the distribution $\boldsymbol{z}$ is sampled from. The latent variable $\boldsymbol{z}$ is typically denoted as the *code*. Figure 2.6b shows how the distributions can be implemented as a CNN. The parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ are then the weights of the different layers, and by training the network on a dataset, the optimal parameters are learned.

The stochastic gradient variational Bayes estimator

A model to sample new data from is desired, so the parameters that maximizes the marginal likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ must be found. Assuming there is some dataset $\mathbf{X} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$ of N i.i.d.

samples of some random variable $\boldsymbol{x}$, the marginal log-likelihood is a sum over each individual log-likelihood for every data point

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(N)}) = \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}). \tag{2.16}$$

As shown in Appendix A.1, each data point can be written

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) = \mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) \middle|\middle| p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})\right) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)}), \tag{2.17}$$

where

$$\mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) \middle|\middle| p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})\right) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) - \log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})\right] \tag{2.18}$$

is the Kullback-Leibler (KL) divergence, measuring the difference between two probability distributions, $\mathbb{E}_q[\cdot]$ is the expectation w.r.t q, and $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)})$ is the variational lower bound on the marginal likelihood. Since the KL divergence by definition is always non-negative, it holds that $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)})$, which is why $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)})$ is called the lower bound. As seen in Appendix A.1, this bound can further be written

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[-\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{z})\right] \\ &= -\mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) \middle|\middle| p_{\boldsymbol{\theta}}(\boldsymbol{z})\right) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\right].\end{aligned} \tag{2.19}$$

The objective is to maximize (2.16), but since the KL divergence in (2.16) involves the unknown $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$, it cannot be maximized directly. Instead, the marginal log-likelihood is estimated by the variational lower bound, as this involves distributions which can be controlled. The distributions $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ must then be more specifically defined.

It is most common to set all distributions to Gaussians, specifically

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \equiv \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}), \boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x})), \tag{2.20a}$$

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}) \equiv \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{2.20b}$$

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) \equiv \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}), \boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z})), \tag{2.20c}$$

where $\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x})$ is diagonal. For a neural network, as in Figure 2.6b, $\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x})$ will map the input to the latent variable's expectation, $\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x})$ will map the input to the latent variable's covariance, $\boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z})$ will map the latent variable to the expected output and $\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z})$ is the output's corresponding covariance, each defined by the network layers. Although the assumptions of (2.20) are simplifying, they allow for efficient computations, and turn out to give good performance. Having defined the distributions, the objective in (2.19) can more explicitly be defined.

The first term to define is the KL divergence. In the general case this must be estimated by e.g. Monte Carlo methods[10], but using the distributions defined in (2.20) it takes the explicit form

$$\mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) \middle|\middle| p_{\boldsymbol{\theta}}(\boldsymbol{z})\right) = \frac{1}{2}\left(\mathrm{tr}(\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})^{\mathsf{T}}\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)}) - k - \log|\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{x}^{(i)})|\right), \tag{2.21}$$

as shown in Appendix A.2, where k is the dimensionality of $\boldsymbol{z}$.

The second term is more problematic, as the expectation is over distributions that are yet to

be parametrized by $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. It could be estimated by averaging $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})$ for a sufficient amount of samples L,

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\right] \approx \frac{1}{L}\sum_{l=1}^{L}\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i,l)}), \tag{2.22}$$

but this is computationally expensive. The expectation is approximated by $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i)})$ instead, where $\boldsymbol{z}^{(i)}$ is a single sample corresponding to $\boldsymbol{x}^{(i)}$. This is usually a poor approximation, but since a batch of data is averaged over when training the model, the approximation holds. Using the distributions defined in (2.20), it takes the form

$$\begin{aligned}\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\right] &\approx \log\mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)}), \boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})\right) \\ &= -\frac{k}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})| - \frac{1}{2}\left\|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})\right\|^2_{\boldsymbol{\Sigma}_{\boldsymbol{x}}},\end{aligned} \tag{2.23}$$

where $\|\cdot\|_{\boldsymbol{\Sigma}}$ is the Mahalanobis norm.

This in total gives the approximation of the variational lower bound,

$$\begin{aligned}\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)}) &= -\mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})\middle\|p_{\boldsymbol{\theta}}(\boldsymbol{z})\right) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i)}) \\ &= -\frac{1}{2}\left(\mathrm{tr}(\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})^{\mathsf{T}}\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)}) - k - \log|\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})|\right) \\ &\quad + \left(-\frac{k}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})| - \frac{1}{2}\left\|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})\right\|^2_{\boldsymbol{\Sigma}_{\boldsymbol{x}}}\right),\end{aligned} \tag{2.24}$$

called the stochastic gradient variational Bayes (SGVB) estimator. As the gradients are usually computed over minibatches $\mathbf{X}^M = \{\boldsymbol{x}^{(i)}\}_{i=1}^M$, where $M \leq N$, the minibatch estimator is

$$\tilde{\mathcal{L}}^M(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathbf{X}^M) = \frac{1}{M}\sum_{i=1}^{M}\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)}). \tag{2.25}$$

The optimal parameters, and thus the distributions, can then be found by maximizing (2.25) for a sufficient amount of minibatches M, as an approximation to (2.16).

A problem when calculating the gradients of (2.25) is that there is a random sampling step in the calculation of $\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)})$, namely sampling $\boldsymbol{z}^{(i)}$ from $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$. This has no gradient, as it is sampled, and the solution is to reparameterize $\boldsymbol{z}^{(i)}$ by the differentiable transformation

$$\boldsymbol{z}^{(i)} = \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)}) + \boldsymbol{\Sigma}_{\boldsymbol{z}}^{-\frac{1}{2}}(\boldsymbol{x}^{(i)}) \cdot \boldsymbol{\epsilon}, \tag{2.26}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The random sampling is now in the auxiliary noise variable $\boldsymbol{\epsilon}$, and the gradient of $\boldsymbol{z}^{(i)}$ can be taken w.r.t. the parameters. Note that (2.26) is just another way of writing $\boldsymbol{z}^{(i)}$ given from (2.20a), as $\mathbb{E}[\boldsymbol{z}^{(i)}] = \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})$ and $\mathrm{Cov}(\boldsymbol{z}^{(i)}) = \boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})$.

The approximation to the log-likelihood (2.16) can finally be solved by finding

$$\begin{aligned}
\boldsymbol{\theta}^*, \boldsymbol{\phi}^* &= \underset{\boldsymbol{\theta},\boldsymbol{\phi}}{\operatorname{argmax}} \frac{1}{M} \sum_{i=1}^{M} \tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)}) \\
&= \underset{\boldsymbol{\theta},\boldsymbol{\phi}}{\operatorname{argmax}} \frac{1}{M} \sum_{i=1}^{M} -\frac{1}{2} \left(\operatorname{tr}(\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})^\mathsf{T} \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)}) - k - \log |\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})| \right) \\
&\qquad + \left(-\frac{1}{2} \log |\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})| - \frac{1}{2} \left\| \boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)}) \right\|^2_{\boldsymbol{\Sigma}_{\boldsymbol{z}}} \right) \\
&= \underset{\boldsymbol{\theta},\boldsymbol{\phi}}{\operatorname{argmin}} \frac{1}{M} \sum_{i=1}^{M} \frac{1}{2} \left(\operatorname{tr}(\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})^\mathsf{T} \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)}) - k - \log |\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})| \right) \\
&\qquad + \left(\frac{1}{2} \log |\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})| + \frac{1}{2} \left\| \boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)}) \right\|^2_{\boldsymbol{\Sigma}_{\boldsymbol{z}}} \right),
\end{aligned} \tag{2.27}$$

where the constant terms in (2.24) have been removed in the optimization. Note that this gives the optimal parameters of the minibatch, which again is an approximation to the whole dataset. Using a stochastic optimization method, e.g. stochastic gradient descent, minibatches from the whole dataset can repeatedly be drawn to find the optimal parameters of the whole dataset.

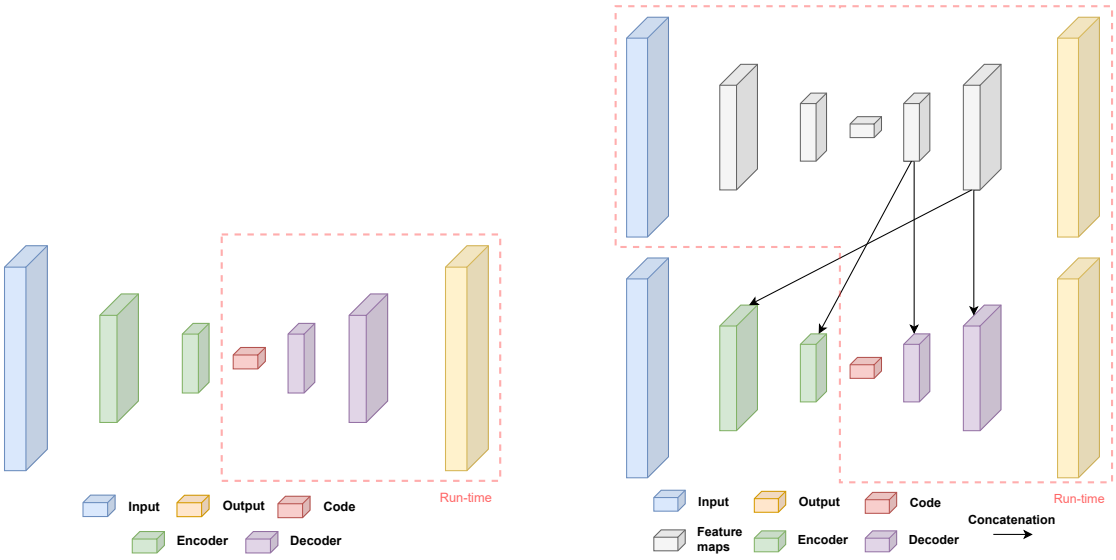
Note also the two terms in (2.27) that are minimized, specifically the KL divergence and the *reconstruction error*. The reconstruction error directly measures how well the model reconstructs the data, which naturally should be as low as possible. In the case of Gaussian distributions, the error is weighted by the uncertainty in the reconstruction, and $\log |\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})|$ naturally acts as a regularizer to ensure the error is not minimized by just driving the uncertainty to large values. The KL divergence, as mentioned previously, is a measure of distance between two distributions, in this case $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{z})$, both given in (2.20). As $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ is fixed to a standard Gaussian and not affected by the changing parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, the KL divergence forces $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ close to a standard Gaussian during the optimization. This is important to ensure that the model is regularized sufficiently so that any code sampled from the latent space is decoded into something meaningful. Otherwise, the reconstruction error would ensure (near) perfect reconstructions of the data, but new data could not be sampled from the code, as the model is only trained to reconstruct the data it has seen. The KL divergence ensures *continuity* and *completeness*, namely that close points in the latent space should give similar output and that points sampled from the latent space should give meaningful output.

The end result is the distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$, which now can be use to sample new data from. The sampling procedure is then:

1. Sample a latent value $\boldsymbol{z}$ from $p_{\boldsymbol{\theta}}(\boldsymbol{z})$.
2. Sample a new data point $\boldsymbol{x}$ from $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$.

Illustrated by the VAE in Figure 2.7a, the generation of new data only involves sampling the code $\boldsymbol{z}$ and decoding it with the decoder $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$. The encoder is therefore not used at run-time.

A common problem when training VAEs using the objective in (2.27) is that the KL divergence is quickly driven towards zero[12]–[14]. This intuitively does not sound like a problem, as the aim, after all, is to minimize the objective, but it heavily regularizes the model from the beginning. The result is a latent representation in which units are inactive during the rest of the training, as they are pruned away before learning a useful representation, with minimum gradient flow between the encoder and decoder. The solution is to penalize the KL term at the



(a) *VAE with blocks used during run-time highlighted.* ***(b)*** *CVAE conditioned on feature maps.*

Figure 2.7: *Possible CNN implementations of a VAE (a) and a CVAE (b).*

start of the optimization, making the model encode as much useful information in $\boldsymbol{z}$ as it can before it is regularized towards the prior, $p_{\boldsymbol{\theta}}(\boldsymbol{z})$. The new objective can be written

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}) = \mathcal{L}_{\text{recon}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}) + \beta \cdot \mathcal{L}_{\text{KL}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}), \tag{2.28}$$

where $\mathcal{L}_{\text{recon}}$ is the reconstruction loss, $\mathcal{L}_{\text{KL}}$ is the KL divergence and β is the weight on the KL divergence. A popular KL annealing schedule is to set $\beta = 0$ for the first couple of epochs, then gradually increase it to $\beta = 1$.

2.2.3 Conditional variational autoencoders

A limitation of the VAE is that there is no control over the data it generates. If it is trained on generating new numbers from the MNIST dataset[15], a random code will give a random digit, which means specific digits cannot be generated. In the case of generating depth maps, it becomes even worse, as a (pseudo) randomly drawn depth map would most likely give no meaningful structure. The conditional variational autoencoder (CVAE)[16] solves this by modeling a *conditional* distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{y})$, conditioned on some $\boldsymbol{y}$. In the case of MNIST, one could condition on the number to be generated, e.g. $p_{\boldsymbol{\theta}}(\boldsymbol{x}|y = 3)$ for generating the number three. For depth maps, it is more complicated, as a depth map should be generated for a given image, so the CVAE needs to condition on this image. Instead of conditioning on the whole image itself, one could condition on its deep features (typically extracted by a CNN), as shown in Figure 2.7b. Here, the conditioning is implemented by concatenation, which effectively gives the network information about what it should reconstruct.

The variational lower bound in (2.19) can be re-written (without individual samples $\boldsymbol{x}^{(i)}$ for notational simplicity) using the conditional distribution as

$$\begin{aligned} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{y}) &\geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) \\ &= -\mathcal{D}_{\text{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y}) \middle|\middle| p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{y})\right) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{y})\right], \end{aligned} \tag{2.29}$$

giving the SGVB estimator

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) = -\frac{1}{2}\left(\operatorname{tr}(\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y})) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y})^{\mathsf{T}}\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y}) - k - \log|\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y})|\right) + \left(-\frac{k}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}, \boldsymbol{y})| - \frac{1}{2}\left\|\boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}, \boldsymbol{y})\right\|^2_{\boldsymbol{\Sigma}_{\boldsymbol{x}}}\right). \tag{2.30}$$

As seen from (2.30), the objective to be optimized for the CVAE is exactly the same as for the VAE in (2.27).

In practice, weighing the reconstruction error by the covariance of the prediction $\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}, \boldsymbol{y})$ is strictly not necessary, but often beneficial. After all, the output of interest for the VAE and CVAE is the reconstruction itself (or rather the newly reconstructed data) and not how uncertain the model is in its reconstruction. This is effectively done by setting $\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}, \boldsymbol{y}) = \mathbf{I}$, giving

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) = -\frac{1}{2}\left(\operatorname{tr}(\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y})) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y})^{\mathsf{T}}\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y}) - k - \log|\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y})|\right) + \left(-\frac{k}{2}\log 2\pi - \frac{1}{2}\left\|\boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}, \boldsymbol{y})\right\|_2^2\right),$$

which results in a mean squared error loss for the reconstruction. This does not require the model to estimate $\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}, \boldsymbol{y})$, but may give poorer results in not doing so. Otherwise, the uncertainty must be estimated same as the reconstruction, and must be an additional output of the model.

CVAEs for depth estimation

Although these models originate and have their main use within the generation of data, recent works have attempted to adapt them to the inference of data. Most notably, [11] used a CNN-based CVAE for predicting depth from individual grayscale images. This work was a step towards a new direction of possible SLAM and 3D reconstruction methods, mainly contributed by the compact representation of depth that the *code* in the CVAE provides. This is effectively a compressed representation of a depth map, with far fewer parameters and has the benefit of allowing joint optimization of pose and map in dense VSLAM and SfM. However, the accuracy in the predicted depth maps, particularly how well the code and decoding capture the full structure, is a limitation of this method. Compared to some of the state-of-the-art methods for monocular depth estimation[17], [18], neither of the CVAE-based methods[11], [19]–[21] seem to produce comparable results, indicating that the code (and how well it is decoded) may be a bottleneck. Research into improving the prediction may therefore be valuable for establishing the CVAE in the fields of VSLAM and 3D reconstruction.

Following the work of [11], a CVAE for depth estimation may be implemented by conditioning on the deep features of the intensity image (e.g. grayscale or RGB). The simplest solution is to extract the first layers of a classification CNN and concatenate them with the features in the encoder and decoder of the CVAE, as shown in Figure 2.8. This architecture consists of two streams: the top stream for extracting features of the input and the bottom stream for encoding and decoding depth conditioned on the features.

Properties for dense SfM and VSLAM

One of the main reasons for using a CVAE for depth estimation is its useful properties in dense SfM or VSLAM. This typically involves the joint optimization of both pose and map, which be-

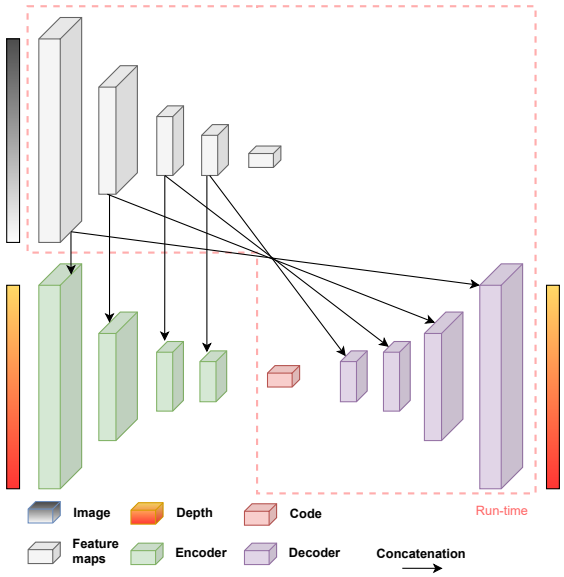


Figure 2.8: *CVAE conditioned on deep features from a contracting CNN.*

comes infeasible (at least in real-time) due to the large number of parameters the dense map presents in the optimization. As shown in [11], the reduced parameter space of the code solves this problem to some extent, making joint optimization of pose and dense map possible. There is still a question of real-time performance and how much detail is lost when using the compact representation of the code.

2.3 Uncertainty estimation

The powerful capabilities of deep learning models often come at the cost of losing explainability, where we as humans have little understanding of how the model reasons or why it performs better or worse for some data. Understanding what it does *not* know is sometimes more valuable than understanding what it does know, especially for safety-critical systems such as the one concerned in this work. Classifying methods often give normalized scores to each class, signaling how confident the model is in its prediction, but this is not the case for regression models, and one could argue it does not necessarily capture model uncertainty even for classification[22]. Bayesian deep learning approaches have recently been introduced to address this problem and separate uncertainty into two types: *epistemic* and *aleatoric* uncertainty. The theory and notation in the following sections are derived from [22].

2.3.1 Epistemic uncertainty

Epistemic uncertainty is the uncertainty of the model itself, that is, the model parameters. It can be referred to as the model uncertainty and can often be explained away given enough data. This makes sense intuitively, as the model should get more confident as it is exposed to more and more samples from the distribution of data.

A prior distribution is put over the model's weights to evaluate the epistemic uncertainty, typically a standard Gaussian $\mathbf{W} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Such models are denoted as Bayesian neural networks, where the otherwise deterministic weights are replaced by distributions which can be averaged

over, instead of directly optimizing the weights. Denoting the input data as $\mathbf{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$, the targets as $\mathbf{Y} = \{\boldsymbol{y}_1, ..., \boldsymbol{y}_N\}$ and the random output of the model as $\mathbf{f}^{\mathbf{W}}(\boldsymbol{x})$, the aim is to maximize the model likelihood $p(\boldsymbol{y}|\mathbf{f}^{\mathbf{W}}(\boldsymbol{x}))$. Given the data, the posterior over the weights $p(\mathbf{W}|\mathbf{X}, \mathbf{Y})$ then captures the set of probable model parameters. As for the variational autoencoder, Bayesian neural networks need variational inference to calculate the otherwise intractable posterior, as

$$p(\mathbf{W}|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \mathbf{W})p(\mathbf{W})}{p(\mathbf{Y}|\mathbf{X})} \tag{2.31}$$

involves the marginal $p(\mathbf{Y}|\mathbf{X})$, which cannot be evaluated analytically. Following variational inference, an approximate distribution $q^*_{\boldsymbol{\theta}}(\mathbf{W})$ is instead fitted to the intractable posterior by minimizing the KL-divergence. The objective to be minimized can then be written

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2\sigma^2}||\boldsymbol{y} - \mathbf{f}^{\widehat{\mathbf{W}}}(\boldsymbol{x})||^2 + \frac{1}{2}\log\sigma^2, \tag{2.32}$$

where $\widehat{\mathbf{W}} \sim q^*_{\boldsymbol{\theta}}(\mathbf{W})$ and σ is homoscedastic aleatoric uncertainty, soon to be further explained. The epistemic uncertainty can finally be found by the predictive variance, approximated by

$$\mathrm{Var}(\boldsymbol{y}) \approx \sigma^2 + \frac{1}{T}\sum_{i=1}^{T} \mathbf{f}^{\widehat{\mathbf{W}}_i}(\boldsymbol{x})^\intercal \mathbf{f}^{\widehat{\mathbf{W}}_i}(\boldsymbol{x}) - \mathbb{E}[\boldsymbol{y}]^\intercal \mathbb{E}[\boldsymbol{y}], \tag{2.33}$$

where T is the number of sampled model weights. The predictive mean is usually modeled by $\mathbb{E}[\boldsymbol{y}] \approx \frac{1}{T}\sum_{i=1}^{T} \mathbf{f}^{\widehat{\mathbf{W}}_i}(\boldsymbol{x})$, such that there is zero parameter uncertainty when all draws $\widehat{\mathbf{W}}_i$ take the same value (as everything except σ^2 cancels).

A problem with calculating epistemic uncertainty is the need for multiple samplings of the model weights, each requiring a full forward through the network. This will inevitably not satisfy real-time constraints in most applications, except for very small models. Epistemic uncertainty is therefore not considered further here.

2.3.2 Aleatoric uncertainty

Where epistemic uncertainty captures uncertainty in the model, aleatoric uncertainty captures it in the observations. Aleatoric uncertainty is further categorized into a *homoscedastic* and *heteroscedastic* part, where homoscedastic uncertainty remains constant for different data and heteroscedastic uncertainty is data-dependent. The latter is of most interest, as it allows the model to reason on different parts of an image in the case of computer vision and assigns higher uncertainty to regions that are likely to produce poor results.

Going back to MAP inference, where single values for the model parameters are found, the aleatoric uncertainty is more easily learned as a function of the data. The objective is in this case

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y}) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{2\sigma(\boldsymbol{x}_i)^2}||\boldsymbol{y}_i - \mathbf{f}(\boldsymbol{x}_i)||^2 + \frac{1}{2}\log\sigma(\boldsymbol{x}_i)^2, \tag{2.34}$$

where $\sigma(\boldsymbol{x}_i)$ is the aleatoric uncertainty for input $\boldsymbol{x}_i$, which is learned as a parameter together with the rest of the model.

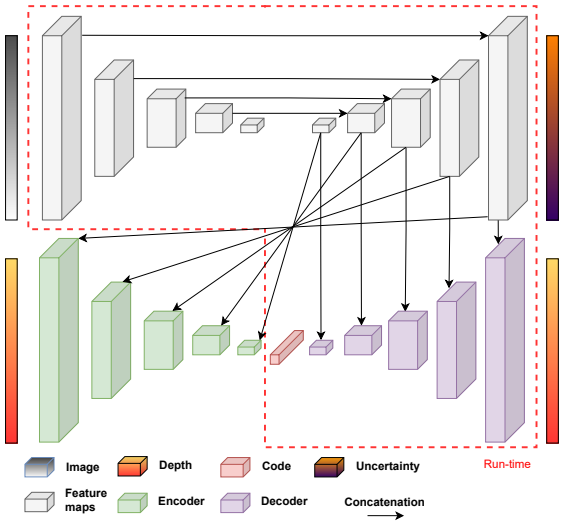


Figure 2.9: *CVAE conditioned on deep features from the expanding path of a U-Net [7], additionally estimating the aleatoric uncertainty of the prediction.*

Aleatoric uncertainty for CVAEs

As the SGVB estimator in (2.30) naturally incorporates uncertainty, the uncertainty of the depth can be estimated as well. This is useful for weighing the reconstruction, as the model will put large weights on areas it struggles to reconstruct and small weights on areas it has no problem with, ultimately letting the model focus on well-posed areas. The uncertainty is also useful for further inferring information about the estimated depth, as it enables the possibility of choosing which estimates to use and which not. As the uncertainty map (uncertainty of each pixel) is a dense prediction, it is typically estimated by an expanding path for enabling fine-grained predictions, as in Figure 2.9.

2.3.3 Confidence propagation

The aforementioned ways of reasoning about a model's uncertainty are rooted in a probabilistic framework. However, the work of [23] is an example of a more engineered way of handling uncertainty. Although this is targeted at sparse data with missing values, the model outputs confidences to densely regressed values and more closely resembles the normalized scores of a classifier.

A confidence map is assigned to the input data, which reflects which values should be trusted and not. For e.g. sparse data, the missing values should have low confidence. This confidence map is then concatenated to the data and fed to the network. A problem with normal convolutions, in this case, is that all channels eventually get mixed, so the confidence is lost in deeper layers. The authors propose normalized convolutions to fix this, which operate on the confidences separately and use confidences continuously in the network to weigh the feature maps. In the final layer, the network outputs a final confidence map with the regressed values, which acts as an uncertainty map. This way of modeling uncertainty has the strength of being robust to missing values but does not have a strict probabilistic interpretation and will therefore not be further considered in this work. It also requires more extensive modifications to the layers to propagate confidences correctly, which was not prioritized.

2.4 Evaluation metrics

When evaluating the quality of a predicted depth map, one can use qualitative and quantitative measures. However, quantitative metrics are most common for comparing methods and serve as the final indication of performance.

A depth map is easily verified qualitatively by humans, as we naturally infer depth from visible light and the semantics of a scene. To some extent, the estimated depth can be verified to be correct, but it is difficult to say to what degree. The fine-grained structure is also difficult to assess from a 2D depth map but much easier from the corresponding 3D point cloud. Therefore, the depth map will be projected out into the world when evaluated qualitatively.

There are several commonly used metrics for evaluating quantitatively, and most research in this area uses multiple different metrics for comparison against state-of-the-art. The most common are:

- Root mean squared error (RMSE) - $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y - y^*)^2}$
- Root mean squared log error (logRMSE) - $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\log y - \log y^*)^2}$
- Absolute relative difference (ARD) - $\frac{1}{N}\sum_{i=1}^{N}|y - y^*|/y^*$
- Squared relative difference (SRD) - $\frac{1}{N}\sum_{i=1}^{N}(y - y^*)^2/y^*$
- Accuracy - $\max(\frac{y}{y^*}, \frac{y^*}{y}) = \delta < \mathrm{thr}$

where y is predicted depth, y^* is ground truth depth and thr is a threshold for accuracy, typically chosen as 1.25, 1.25^2 and 1.25^3. Note that lower values indicate better performance for all metrics except for accuracy. When comparing different models, it is useful to have some overall performance metric. This work proposes a new metric, the *average performance score* (APS), which scores a method relative to the best performing models for each metric, giving the relative overall performance. The APS is defined as

$$\text{APS} = \frac{1}{M_E}\sum_{i=1}^{M_E}\frac{m_i}{m_{i,min}} + \frac{1}{M_A}\sum_{i=1}^{M_A}\frac{m_{i,max}}{m_i}, \tag{2.35}$$

where M_E is the number of error metrics used and $m_{i,min}$ is the value of the best performing model for the specific error metric, and M_A is the number of accuracy metrics used and $m_{i,max}$ is the value of the best performing model for the specific accuracy metric. The APS is defined in the range [1, ∞), where an APS of 1 is the best possible.

2.5 Related work

Depth estimation from images has been a heavily researched topic for numerous years, early on, mainly by inferring the scene's structure from the motion of the camera or using known relative poses between the cameras. In the last decade, methods based on deep learning have shown promising results in a large variety of computer vision-related tasks, with depth estimation

among them. Monocular depth estimation has been of most significant focus, as estimating depth from individual images is ill-posed and impossible without geometric priors of the scenes[2] and is also the focus of this work. Therefore, the reviewed and related work is limited to monocular methods that utilize deep learning to estimate depth.

Eigen *et al.*[8] were among the first to use a pure CNN-based architecture for depth estimation. Global context and cues for a single image were extracted by a deep contracting path, giving a coarse prediction of the depth in the scene, which was then fused with a finer, higher-resolution path to refine the coarse depth map into a fine-grained map that preserves local details. The method achieved state-of-the-art performance on popular datasets and set the foundation for later CNN-based depth estimation methods to come. The key contribution is perhaps how they preserved local details with global context by concatenating the coarse depth map with finer feature maps, sharing the same conceptual idea as the later introduced and popularly used U-Net[7].

Laina *et al.*[9] used the residual connections of [6] to predict depth from a very deep CNN, using the typical contracting path for calculating deep features, and a subsequent expanding path with upsampling to get a dense map of higher resolution. These autoencoders and are frequently seen in dense prediction tasks such as depth estimation or segmentation.

In Wofk *et al.*[24], a combination of the feature concatenation in [8] and autoencoder structure of [9] was used, with a focus on achieving real-time performance on smaller platforms. Computational complexity was lowered by fusing feature maps through addition rather than concatenation and pruning away redundant parameters.

Different from the previously described methods, Ranftl *et al.*[17] used a Transformer for predicting depth, relying on the attention-mechanism instead of convolutions to extract global context. The architecture was, however, similar to an autoencoder and fused features from the encoder with convolutional layers in the decoder, same as e.g. [24]. The model outperformed all others on the popular datasets and is one of the current state-of-the-art methods for single-view depth estimation.

Generative models have also shown competing performance in inference tasks such as depth estimation, even though they are primarily targeted at data generation. Bloesch *et al.*[11] used a variational autoencoder conditioned on the deep features of the images, generating depth maps from a compact latent representation. An essential contribution is the latent representation itself, being a compressed and lower-dimensional representation of the depth maps, which exhibits beneficial properties for further dense 3D reconstruction or VSLAM.

The works described have predicted depth from individual images and have not used information from multiple views. In Godard *et al.*[25], temporally adjacent frames were used to train an autoencoder in a self-supervised manner, using photometric errors as a loss function rather than ground truth depth values. Similarly, in Yang *et al.*[26], two subsequent autoencoders were used to predict the disparity maps instead of depth directly. By training on stereo images, a virtual stereo term was adopted to improve the accuracy of monocular odometry.

Spatiotemporal networks (sometimes referred to as multi-view stereo) have recently shown promise for monocular depth estimation from videos, where Yao *et al.*[27] used a 3D convolutional network to extract both spatial and temporal features from multiple views. Although computationally expensive, these networks are among the current state-of-the-art.

3 | Method

This chapter introduces the chosen deep learning architecture and methods used for monocular depth estimation. The methods are divided into two main categories: single-view and multi-view methods. Although many modifications to the baseline architecture are presented for the single-view case in Chapter 3.2, they are just as applicable to the multi-view setting. The goal of this work is not to find a best-performing network with fine-tuned hyperparameters and the like but rather to experiment with more extensive changes and how they affect the performance for depth estimation. The focus here is, therefore, on the modifications themselves.

3.1 Architecture

A CVAE based on [11] was chosen as a baseline for the depth estimation network, with background in the authors previous work[1]. The network, titled Depth-CVAE, has its general structure seen in Figure 3.1, but a more detailed description is given in Appendix B. The Depth-CVAE consists of two main streams: a U-Net in the top stream for predicting the uncertainty in depth estimates and a CVAE conditioned on the deep features of the U-Net in the bottom stream for predicting depth.

The Depth-CVAE was implemented[1] in PyTorch[28] using the framework of [29], and with general architecture based on [11]. The encoder of the top stream is a ResNet-18[6] without the final classifying layers, and each layer in the rest of the network adopts the same residual connections for improving gradient flow. Fine-grained information is retained by concatenating higher-dimensional feature maps between the encoder and decoder of the U-Net, and similarly between the decoder of the U-Net and the CVAE.

Loss function

The traditional noise model in the framework of variational inference is Gaussian, but the Depth-CVAE uses a Laplace distribution for modeling the noise,

$$p(x) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right), \tag{3.1}$$

where $b > 0$ is referred to as the diversity of the distribution but in practicality is a measure of uncertainty, and plays the exact same role as $\boldsymbol{\Sigma}$ in the Gaussian case. The resulting reconstruction error is now an ℓ_1 loss, instead of ℓ_2, which has been shown to produce better results in computer vision related deep learning problems concerning depth[9]. The final loss function can then be

[1] `https://github.com/andersfagerli/Depth-CVAE`

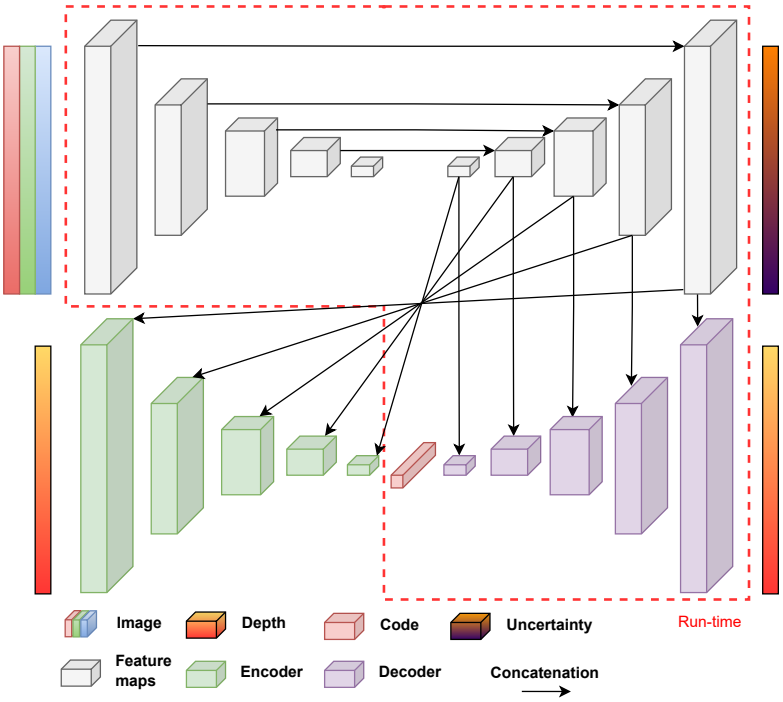


Figure 3.1: *Graphical structure of the Depth-CVAE, with a U-Net in the top stream for predicting uncertainty and a CVAE in the bottom stream for predicting depth. The input to the network is in this case a single RGB image, with corresponding depth during training.*

written as

$$\begin{aligned}\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) &= \mathcal{L}_{\text{recon}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) + \beta \cdot \mathcal{L}_{\text{KL}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) \\ &= \frac{1}{|\Omega|} \sum_{\boldsymbol{u} \in \Omega} \frac{|\boldsymbol{x}(\boldsymbol{u}) - \boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{u})|}{B(\boldsymbol{u})} + \log\left(2B(\boldsymbol{u})\right) \\ &\quad + \beta \cdot \frac{1}{2}\left(\text{tr}(\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y}) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y})^{\mathsf{T}} \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y}) - k - \log|\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y}|)\right), \end{aligned} \tag{3.2}$$

where $\boldsymbol{x}$ is the ground truth depth map, $\boldsymbol{y}$ is the image, B is the uncertainty map and $\boldsymbol{u}$ are the pixel coordinates in the set of all pixels Ω with cardinality $|\Omega|$.

Code

To reduce the number of parameters in the network, it is common to let $\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x})$ and $\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x})$ share layers. This is typically done by sharing all the convolutional layers before two separate fully connected layers are used for each of them. A possible implementation of the code block is shown in Figure 3.2. The covariance matrix is assumed to be diagonal, such that each element of $\boldsymbol{\Sigma}_{\boldsymbol{z}}$ in Figure 3.2 corresponds to each element in the diagonal.

The code is an encoded version of the ground truth depth image during training, but at run-time, when the model predicts depth from an image alone, the code must be sampled from the prior, $p_{\boldsymbol{\theta}}(\boldsymbol{z})$. As the prior is a standard Gaussian, it has a distinct peak at its expectation, called the *zero-code*. Based on all training data, this geometric prior is the most expected to observe and is therefore used at run-time. A code is thus not sampled but just chosen as $\boldsymbol{z} = \mathbf{0}$.

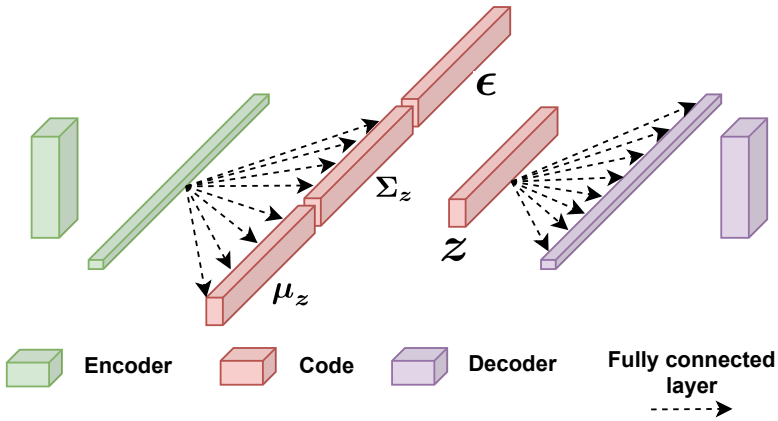


Figure 3.2: *Implementation of the code using neural networks. The final layer of the encoder is flattened and forwarded through two separate fully connected layers that each make up the expectation and covariance of the code distribution. The code $\mathbf{z}$ is then sampled using the reparameterization* (2.26)*, and forwarded through one fully connected layer to generate the first decoder layer.*

3.2 Single-view depth estimation

Estimating depth from single views is the most heavily researched branch of depth estimation within deep learning, much due to the ill-posed problem with traditional methods. This chapter presents modifications and suggestions for further improving the performance of the chosen architecture, solely relying on single views.

3.2.1 Baseline

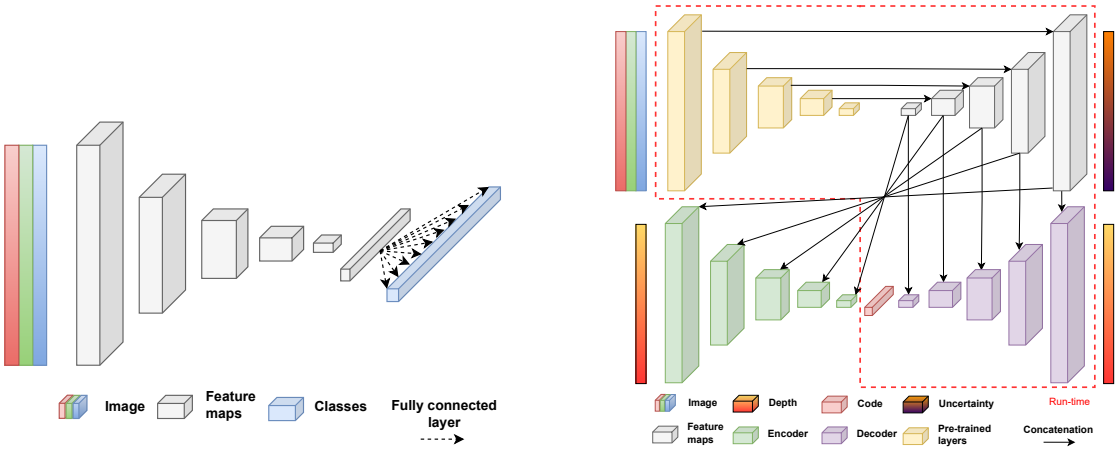
The network displayed in Figure 3.1 and detailed in Appendix B serves as a baseline for comparison against experiments with further modifications. Despite being a straightforward CVAE, the number of parameters in itself should give a good performing depth estimation network, with a final count of 72M parameters. Excluding the encoder of the bottom stream, which is not used at run-time, the network has 46M parameters. This baseline will remain relatively unchanged in further experiments, such that the modifications themselves will indicate worse or better performance.

RGB vs. grayscale images

A question going forward is whether to use RGB images or only their luminance component as input to the network, essentially questioning the importance of *color* for reasoning on the depth in a scene. It is known from image and video compression[30] that most information in color images is contained in the luminance component, being a linear combination of the RGB channels,

$$Y = 0.299R + 0.587G + 0.114B, \tag{3.3}$$

and that it alone explains most variability in the data. A benefit of only using this component is reducing input channels to the network, giving fewer parameters and faster inference while possibly giving the same performance. In fact, [11] observe an insignificant decrease in accuracy in their depth estimates when converting to grayscale, albeit on very different data than what is considered in this work. Both RGB and grayscale images will therefore be considered in the experiments.



(a) *ResNet-18 trained to classify images on the ImageNet dataset.*

(b) *Depth-CVAE with a pre-trained encoder in the top stream.*

Figure 3.3: *The ResNet-18 pre-trained on ImageNet (left) and its corresponding layers highlighted in yellow in the Depth-CVAE (right).*

3.2.2 Transfer learning

A common technique for boosting the performance of a machine learning task is to transfer knowledge from a related domain into the targeted domain. This is especially the case if there is a large amount of data in the related domain but not in the targeted domain, enabling the model to generalize better. Knowledge is, in this case, the weights of the network, which by transfer learning are directly transferred from the pre-trained network to the targeted network. Although the end performance might not always be boosted, the training is typically much easier, starting at a lower loss and with a steeper slope.

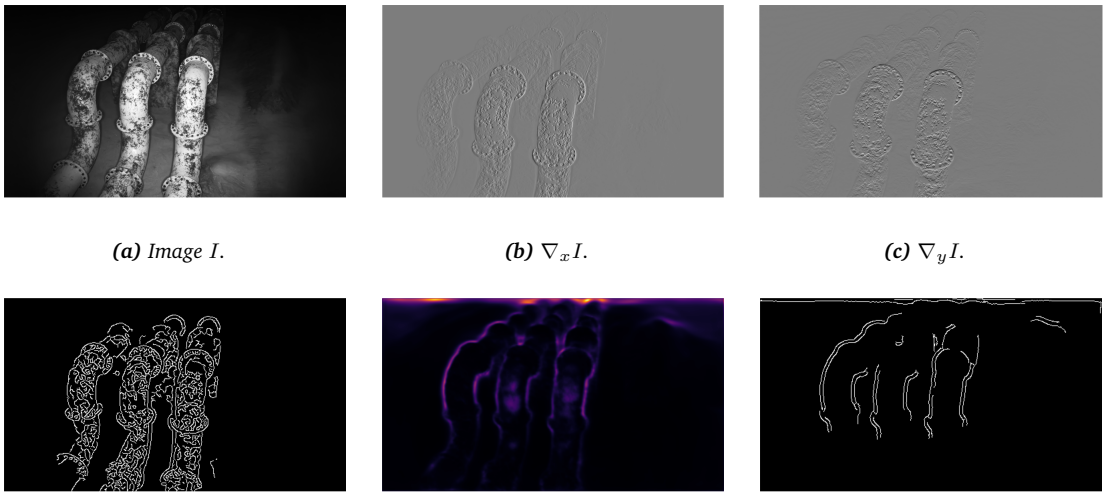
ImageNet transfer learning

As deep learning models related to computer vision often learn features in images, regardless of what the network is predicting, it is common to use a network trained on a large corpus of images, typically the ImageNet classification dataset[31], consisting of 3.2M images. Although this pre-trained network is trained for classification, many of the layers will possibly share knowledge in the domain of depth estimation, specifically early in the pipeline with layers close to the image. Only the encoder of the top stream of the Depth-CVAE is therefore initialized with pre-trained weights, as shown in Figure 3.3. As the ResNet-18[2] used for transfer learning is trained on RGB images, the model must also use RGB images as input.

SceneNet RGB-D transfer learning

A limitation with pre-training on ImageNet is that the data does not directly relate to depth. Therefore, the transferred knowledge may only be used in the early stages of the pipeline. The model should not only generalize well to different features and cues in images but also the geometry of different scenes and objects. SceneNet RGB-D[32] is a synthetic but photorealistic dataset consisting of 5M RGB-D images from indoor scenes with various objects. Since each image has an accompanying depth map, a model can be trained to predict depth on this large dataset and later transfer its knowledge to a targeted domain. A question is if the scenes and

[2]The pre-trained model is downloaded from `https://pytorch.org/vision/stable/models`.



(a) *Image I.* ***(b)*** *$\nabla_x I$.* ***(c)*** *$\nabla_y I$.*

(d) *Canny edges.* ***(e)*** *Aleatoric uncertainty.* ***(f)*** *Canny edges on uncertainty map.*

Figure 3.4: *Top row: A grayscale underwater image (left) with gradients along x-direction (middle) and y-direction (right). Gradients are approximated by finite differences. Bottom row: Canny edges detected on I (left), aleatoric uncertainty of model on I (middle) and canny edges detected on uncertainty map (right).*

objects are relatable to the target domain and if the difference in scale poses difficulties in transferring knowledge, which may give even poorer convergence than when training without transfer learning.

3.2.3 Uncertainty-aware smoothness

The local geometry of most scenes in the world is typically smooth, with neighboring points highly correlated. This is a geometric prior one can take advantage of to aid the model in predicting the geometry of an image, where one can enforce the model to make locally smooth predictions. This is, of course, not always the case, and this prior fails especially at edges, where there are depth discontinuities. A commonly used regularizer is the edge-aware smoothness [18], [26], [33],

$$\mathcal{L}_{\text{smooth}}(D, I) = \frac{1}{|\Omega|} \sum_{\boldsymbol{u} \in \Omega} |\nabla_x D(\boldsymbol{u})| e^{-|\nabla_x I(\boldsymbol{u})|} + |\nabla_y D(\boldsymbol{u})| e^{-|\nabla_y I(\boldsymbol{u})|}, \tag{3.4}$$

where D is a depth map and I is the corresponding image. This regularizer penalizes areas with a low image gradient, corresponding to areas that should be locally smooth and down-weighs areas with a high image gradient. As edges usually have more significant image gradients, this regularizer will approximate the edges by their gradients and, in theory, be aware of them. This will, however, also down-weigh areas that are locally smooth but still have high gradients, typically on texture-rich surfaces. This is typical for some underwater scenes, as seen in the top row of Figure 3.4, where texture is even more significant than edges, and the prior in (3.4) may therefore fail in these cases.

Instead of approximating edges by their gradients, an edge detector may be used on the image. The Canny edge detector[34] is an old but popular edge detector, and applying it to Figure 3.4a results in Figure 3.4d. Even though the image is smoothed by a Gaussian filter before passing it to the edge detector, it is clear that many non-edges get detected as edges,

same as using image gradients, and canny edges are therefore not suited.

An observation from the results of the network on underwater data is that the uncertainty of the model is high around edges and not so much on smooth surfaces, as seen in Figure 3.4e. Leaving further discussion on the resulting uncertainty to Chapter 4, the takeaway here is that the uncertainty map might be suited for edge detection, as the uncertainty along edges is sharp while the uncertainty at surfaces is smooth. Applying the Canny edge detector on a smoothed uncertainty map will then result in detected edges, where non-edges are filtered out because they do not have high enough image gradients in the uncertainty map. The result can be seen in Figure 3.4f, where only edges are retained. As the Canny edge detector outputs binary values (0 for non-edges and 1 for edges), the prior is changed to

$$\mathcal{L}_{\text{smooth}}(D, B) = \frac{1}{|\Omega|} \sum_{\boldsymbol{u} \in \Omega} |\nabla_x D(\boldsymbol{u})|(1 - E(B(\boldsymbol{u}))) + |\nabla_y D(\boldsymbol{u})|(1 - E(B(\boldsymbol{u}))), \tag{3.5}$$

here called the *uncertainty-aware smoothness*, where B is the uncertainty map, and E is the edge-detector applied on B.

A possible problem with the uncertainty-aware smoothness is that not all edges are detected. The prior in (3.5) will ensure smooth geometry on all parts that do not contain edges, so non-smooth parts of the scene will possibly be made smooth if their corresponding edges are not detected. Also, as seen in Figure 3.4f, each edge is detected twice. This is because the uncertainty at edges is more than one pixel wide, therefore having large gradients on each side. The biggest disadvantage is that the detected edges are dependent on the estimated uncertainty of the model, meaning that few edges will be detected if the model estimates low uncertainty, and many non-edges will be detected if there is high uncertainty. The uncertainty-aware smoothness loss should therefore not be active until the model outputs consistent uncertainty maps, typically after several epochs.

The final loss function with edge-aware smoothness or uncertainty-aware smoothness is

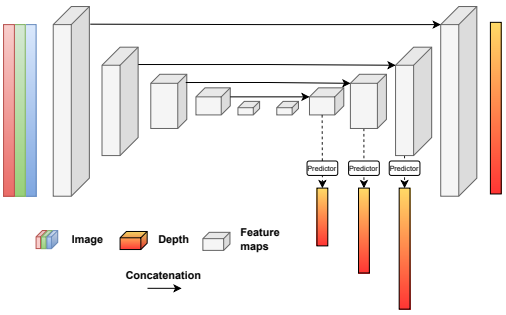
$$\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) = \mathcal{L}_{\text{recon}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) + \beta \cdot \mathcal{L}_{\text{KL}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) + \lambda \cdot \mathcal{L}_{\text{smooth}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}), \tag{3.6}$$

where λ is a penalty on the smoothness regularizer.

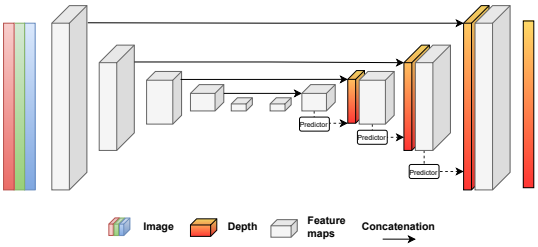
3.2.4 Multi-scale loss

It is common to evaluate the loss at multiple scales to aid deep learning models in gradually constructing the output prediction. Similar to [35], predictor heads are placed at multiple feature maps to make predictions from layers with lower resolution, which then contributes to the overall loss. The goal is to make each layer learn weights that better relate to depth, not just the final layer, constructing gradually from coarse to fine-grained depth maps. The number of parameters in the predictor heads should, therefore, be as small as possible to ensure that the heads solely are not learning to construct depth. The predictors will typically consist of a convolution for merging channels, upsampling for matching the target size, and activation for the final prediction.

Along the lines of [18], Figure 3.5a shows one possible implementation, where predictors are placed at several layers and output a depth map at each resolution. The loss is then calculated and averaged for each prediction. Although many use 3×3 filters when merging channels, this work will use 1×1 filters as a bottleneck to ensure as few parameters as possible. The depth maps are then upsampled to the original image size before being passed through a sigmoid activation.



(a) *Multi-scale loss with predictions decoupled from subsequent layers.*



(b) *Multi-scale loss with predictions concatenated with subsequent layers.*

Figure 3.5: *Two possible ways of implementing multi-scale loss, illustrated with a U-Net for simplicity.*

Figure 3.5b shows another possible implementation, e.g. used in [8] and [36], where the multi-scale predictions are subsequently concatenated in the following layer, more tightly integrating depth into the pipeline. This will, however, also increase the number of parameters and inference time, in contrast to Figure 3.5a which does not add overhead during inference.

Evaluating the loss at multiple scales, the loss function becomes

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) = \frac{1}{S} \sum_{s=1}^{S} \mathcal{L}^s_{\text{recon}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) + \beta \cdot \mathcal{L}_{\text{KL}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}), \tag{3.7}$$

where S is the number of output scales. A smoothness prior can also be applied on each output scale, but should not penalize the lower scales as heavily, as these are coarse. Following [18], the loss function at multiple scales with smoothness regularization is

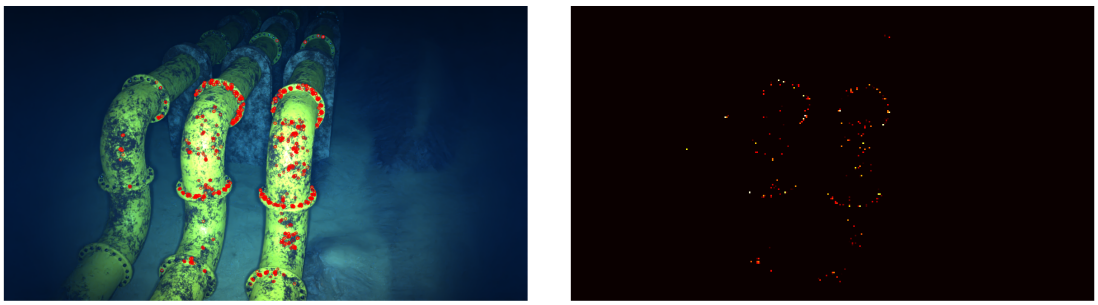
$$\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) = \frac{1}{S} \sum_{s=1}^{S} \mathcal{L}^s_{\text{recon}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) + \lambda^s \cdot \mathcal{L}^s_{\text{smooth}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) + \beta \cdot \mathcal{L}_{\text{KL}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}), \tag{3.8}$$

where $\lambda^s = 10^{-3} \cdot \frac{1}{2^{s-1}}$.

3.2.5 Depth in the top stream

As the CVAE in Figure 3.1 concatenates feature maps from the top stream with its own, it is said to *condition* on these feature maps. In this way, the conditioning variables provide information to aid the network in its prediction, and variables better relating to what the model should predict will perhaps condition it better. Since the top stream predicts uncertainty, the bottom stream is conditioned on feature maps relating to uncertainty, which may not be optimal for predicting the depth of a scene.

As an experiment, the top stream is changed to predict depth, and the bottom stream is changed to predict both depth and uncertainty. Predicting depth in both the top and bottom stream may seem superfluous, but the goal is to condition the bottom stream better by feeding feature maps that relate more strongly to depth. Additionally, a possible weakness with the CVAE in [11] is that uncertainty is predicted before depth and therefore not tightly coupled with it the entire way. The SGVB estimator in (2.30) naturally incorporates uncertainty at the very end with the final prediction, so estimating uncertainty together with depth follows the more probabilistic framework of variational autoencoders.



(a) *ORB features detected and marked in red.* ***(b)*** *Sparse depth at corresponding ORB feature locations.*

Figure 3.6: *Detected ORB features (left) and their corresponding depth values (right).*

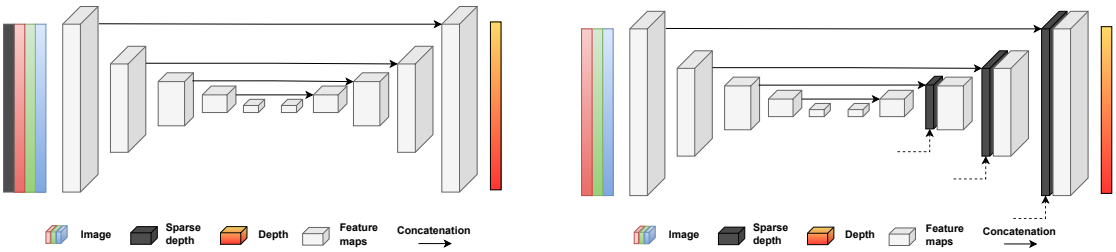
3.2.6 Auxiliary sparse depth

Using information from images alone is not the only way of reasoning about the depth of a scene. Other sensors may be readily available, which signals serve as auxiliary input to the model. Distance-sensing sensors are especially useful in this case, as they directly measure what the network aims to model. However, most of these sensors fail underwater, so using depth measurements from e.g. LiDAR and RGB-D cameras will not be possible. Acquiring depth estimates from some other computer vision-based method is a possibility, but these estimates should be close to the ground truth. Today, and especially underwater, this accuracy is usually obtained by feature-based methods, which provide a sparse depth map. For example ORB-SLAM3[37] computes depth at ORB features, giving a sparse geometric reconstruction of the scene, as seen in Figure 3.6.

In [38] a sparse depth map is concatenated with its corresponding RGB image to add auxiliary data for the prediction, improving over the baseline with RGB only. This is a straightforward way of adding the sparse data, but it is not optimal. The sparse depth map contains values at a sparse set of locations, and all other values are initialized to zero. As zero corresponds to some depth, the invalid values will falsely indicate some structure at these locations, typically very close or very far away depending on the chosen depth parameterization. If the sparse depth map is not too sparse, one can e.g. average out values in a close neighborhood to a missing value to fill out the entire map, but this is still a poor solution for very sparse maps, such as the ones typically obtained by ORB-SLAM3.

An approach to solve this issue is to assign confidences to each pixel in the sparse depth map and propagate these confidences throughout the entire network, as in [23]. Pixels containing values get initialized with 100% confidence, and pixels containing no values (zeros) get initialized with 0% confidence. The network then outputs a confidence map with the prediction that displays its uncertainty in each prediction, similar to [22], but without the probabilistic interpretation. This, however, requires significant changes to the network layers and adds an additional channel to the input. The solution is therefore not further investigated here.

Following [38], and more notably [20] and [21], the sparse depth map is concatenated to the RGB image to form a 4-channel input to the network, as seen in Figure 3.7a. Depth values are sampled from the ground truth depth map at detected ORB features, and all other pixels are set to zero. This will provide unrealistically good sparse depth to the network, as ORB-SLAM3 or any other method cannot match the ground truth in accuracy, and a better approach is to add noise



(a) *Sparse depth concatenated at input.*

(b) *Sparse depth concatenated at multiple scales in decoder.*

Figure 3.7: *Two possible ways of adding auxiliary sparse depth, illustrated with a U-Net for simplicity.*

to the samples. However, the goal here is to validate how auxiliary data affects performance, so noise will not be added here.

A possible problem with adding sparse depth to the input, as in Figure 3.7a is that two significantly different types of data are mixed early on. It is difficult to say if this is a problem, but as a hypothesis, it may be better to add sparse depth later in the pipeline, where the network is learning to construct depth. In Figure 3.7b, the sparse depth map is resized at multiple scales and concatenated at multiple layers in the decoder. This ensures that the sparse depth is mixed with features more closely related to depth, possibly aiding the network further in its reconstruction. However, the poor solution of assigning zeroes to missing values is possibly even worse here, as the sparse depth maps are added closer to the final prediction.

3.2.7 Data augmentation

Data augmentation is central to any machine learning task, where e.g. horizontal and vertical flips, crops, rotations and color transformations are common within computer vision. This is especially relevant for datasets with fewer data and may significantly boost the performance on the test set. However, for the dataset in this work, one could argue that some common augmentations will give a false indication of performance on other data. This will be further explained when the dataset is presented and data augmentations are not used for now. Additionally, since data augmentation is an established way to boost a model's performance, it is not of interest in this work, as the only goal is to investigate how new modifications may work.

3.3 Temporal multi-view depth estimation

As estimating depth in scenes with traditional model-based methods requires multiple views, one would also expect deep learning models to benefit from multiple views. An important remark is that this work only considers monocular setups, and the proposed methods will therefore only rely on temporally adjacent images collected by a single moving camera. These networks are so-called *spatiotemporal* deep learning models.

3.3.1 Baseline

The baseline for the experiments in the multi-view setting can be seen in Figure 3.8, where three consecutive grayscale images are concatenated at the input, together with their corresponding depth maps at the input of the CVAE. The network predicts each image's uncertainty and depth map, thus making predictions for all images simultaneously.

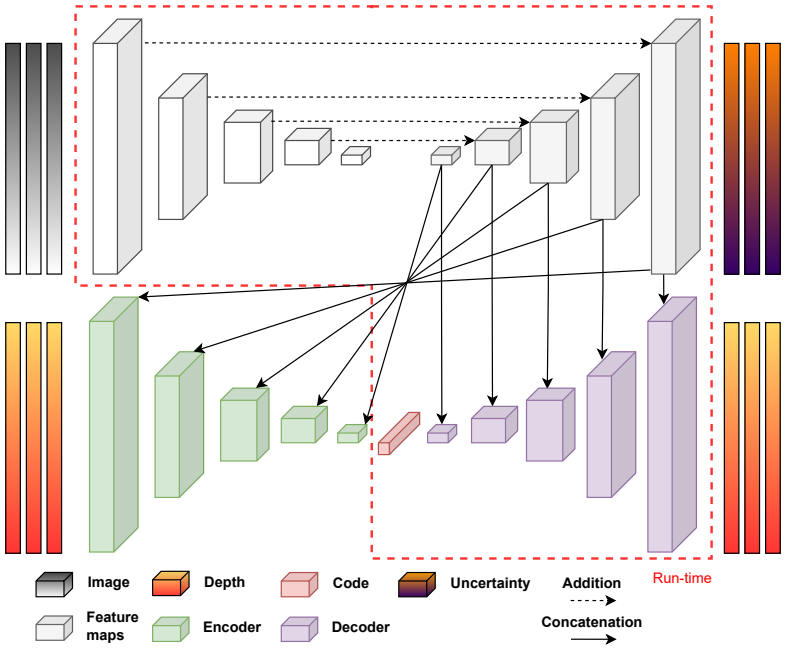
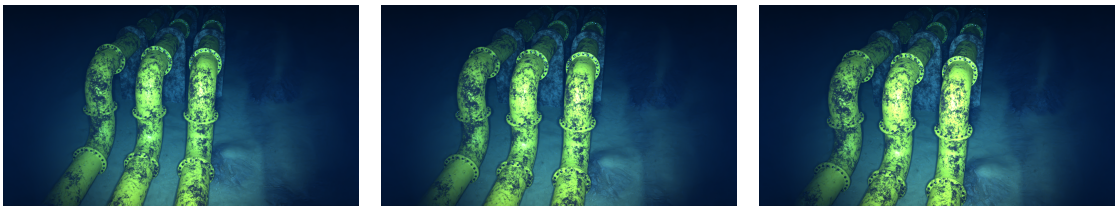


Figure 3.8: *Baseline architecture for the multi-view Depth-CVAE.*

The architecture for the baseline network is roughly the same as for the single-view case. The input and output dimensionality is increased, and the top stream uses skip connections instead of concatenations to reduce the number of parameters. Although the architecture is conceptually simple, it may not be optimal for regressing on multiple views, which may require a completely different architecture to handle the increased complexity. The scope of this work will, however, limit itself to variations of the baseline in Figure 3.1, where the goal is to see if using multiple views in some way can boost the performance.

Because the dimensionality at the input is increased, the number of parameters and computational cost also increases. In order to satisfy computational constraints on the experimental platform, RGB images can no longer be used without further modifications to the architecture, and grayscale images are therefore used instead. Note that this in itself may affect the performance of the network.

Another consideration is the extent of motion between the consecutive images. In traditional SfM or VSLAM, adequate motion parallax is crucial for the performance of the method, where too little motion between keyframes renders the baseline too small, e.g. for estimating depth correctly, and too much motion results in large occlusions and few overlapping areas. Ideally, one should use a keyframe-based approach for choosing the consecutive images, such as the covisibility criteria in ORB-SLAM, but this is difficult to implement with the data loaders in PyTorch. Instead, a chosen fixed number of frames are skipped between images in the dataset, where the number of skipped images is chosen based on manual inspection of the consecutive images in the dataset. Every fourth frame is chosen for the VAROS dataset, as this should give enough motion, with an example seen in Figure 3.9. Note that this way of choosing frames is heavily dependent on the motion of the camera and will fail when the camera is undergoing slow or fast movements.



(a) *First image.* ***(b)*** *Middle image.* ***(c)*** *Last image.*

Figure 3.9: *Three consecutive images taken from the VAROS dataset, skipping three inbetween images to ensure enough motion.*

3.3.2 Motion filter pre-training

As multiple images are given as input, the feature maps close to the input will no longer learn the features of a single image but rather correlations between the images. It is difficult to say what filters are learned, as they do not give the same visual cues as e.g. the edge filters for single images. However, since the motion between the images is essentially the difference from the multi-view to the single-view case, the filters will in some way be related to motion. Using e.g. ImageNet for transfer learning is therefore no longer possible since the learned features from ImageNet are not related to motion.

The proposed solution is to train a multi-view network on a large dataset for a task related to motion and transfer parts of the learned features related to the target task. Training a network to estimate depth from multiple images is one possible way, but this restricts the size of the pre-training dataset to sequences that have ground truth depth, which is typically hard to come by (except for synthetic datasets), and the resulting pre-trained network will see a limited amount of data.

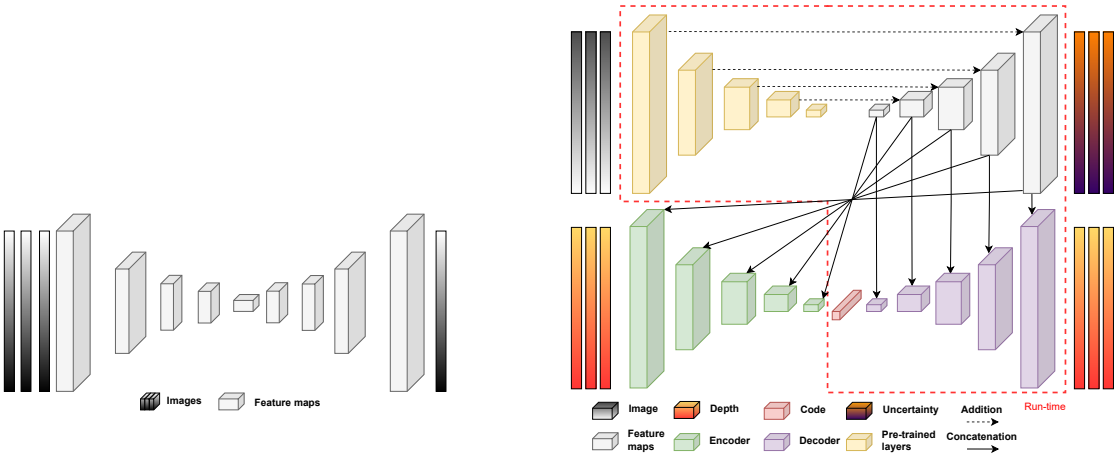
Instead, a plain autoencoder is trained to predict the next image from a set of consecutive images, as displayed in Figure 3.10a. By predicting the next consecutive image, the model is forced to use the motion between the previous images to aid its reconstruction of the following image and thereby learns motion filters in its encoder. The strength of this method is that it is *self-supervised*, as the only necessary data are sequential images. This way, any video can be used to train motion filters, and the dataset is practically unlimited in size. Only the KITTI raw dataset[39] is used to illustrate the motion pre-training here. The network uses a mean squared error loss as the objective function,

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{y}) = \frac{1}{|\Omega|} \sum_{\boldsymbol{u} \in \Omega} (\boldsymbol{y}(\boldsymbol{u}) - \hat{\boldsymbol{y}}(\boldsymbol{u}))^2, \tag{3.9}$$

where $\boldsymbol{y}$ is the ground truth next image and $\hat{\boldsymbol{y}}$ is the predicted next image, and the detailed network architecture is given in Appendix C. After the network is pre-trained on KITTI, the weights are transferred to the multi-view Depth-CVAE, as shown in Figure 3.10b. As for the ImageNet transfer learning in the single-view case, only the feature maps close to the input are transferred since the decoders of the two networks do not share knowledge due to their different output types.

3.3.3 Multi-view consistency

A possible problem with the baseline architecture is that the model is in no way forced to interconnect information from the different inputs. In the worst case, each of the three consecutive



(a) *Autoencoder trained on predicting the next image from a set of consecutive images.*

(b) *Multi-view Depth-CVAE with a pre-trained encoder in the top stream.*

Figure 3.10: *Autoencoder pre-trained on KITTI (left) and its corresponding layers highlighted in yellow in the multi-view Depth-CVAE.*

images are forwarded independently to produce the output, effectively using one-third of the parameters compared to the single-view case. Instead of assuming a learned linkage between the images, one can enforce this by producing consistent depth estimates between multiple views. This is similar to comparing photometric values between overlapping images, under the assumption of no photometric and geometric distortions, where the intensity value of one pixel should be the same as the corresponding pixel in another image with an overlapping view.

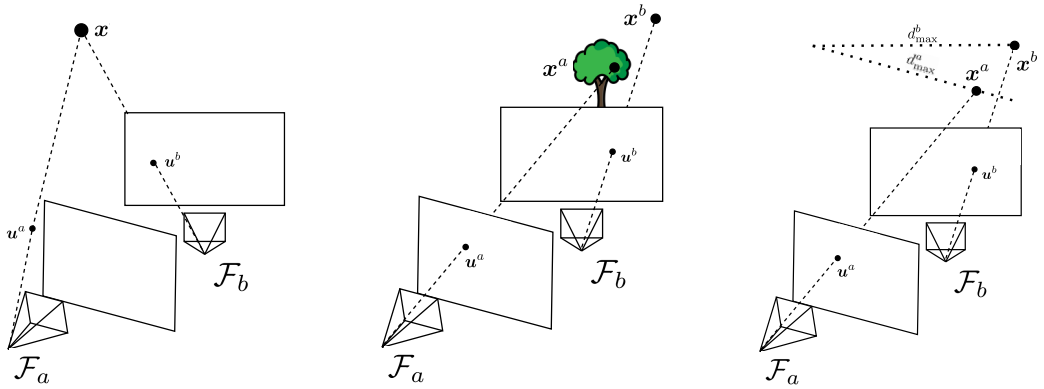
Given two cameras $\mathcal{F}_a$ and $\mathcal{F}_b$ and the relative pose between them $\mathbf{T}_{ab}$, the *geometric error* from $\mathcal{F}_b$ to $\mathcal{F}_a$ at a given pixel $\boldsymbol{u}^b$ is

$$e_g(\boldsymbol{u}^b, z^b) = \left|\mathbf{T}_{ab}\pi_p^{-1}(\boldsymbol{u}^b, z^b))\right|_z - D_a(w(\boldsymbol{u}^b, z^b, \mathbf{T}_{ab})), \tag{3.10}$$

where $z^b \equiv D_b(\boldsymbol{u}^b)$ is the depth in view $\mathcal{F}_b$, $\pi_p^{-1}(\cdot)$ is the perspective camera backprojection (2.10), $w(\cdot)$ is the warp function (2.11), D_a is the depth map of view $\mathcal{F}_a$ and $|\boldsymbol{x}|_z$ takes the z-component of $\boldsymbol{x}$. In more practical terms, the depth value at $D_b(\boldsymbol{u}^b)$ is backprojected into the 3D point $\boldsymbol{x}^b$ and transformed into view $\mathcal{F}_a$ by $\mathbf{T}_{ab}$, where it should have identical depth to the corresponding pixel in D_a. Notice that the relative pose is assumed to be known here.

Using (3.10) in the loss function, the network is forced to predict depths that are consistent with corresponding points in consecutive depth maps and will therefore depend more on using information from several images to reduce the geometric error. Since the geometric error has a metric scale, in contrast to the network predictions, which lie in the range [0, 1] due to the sigmoid activation in the final layer, the geometric errors are normalized before being passed to the loss function. Denoting $D_a^{'}(\boldsymbol{u}^a) = \left|\mathbf{T}_{ab}\pi_p^{-1}(\boldsymbol{u}^b, z^b))\right|_z$, the geometric loss with normalization as in [40] is given as

$$\mathcal{L}_{\text{geometric}}(D_a, D_b) = \frac{1}{|\Omega|}\sum_{\boldsymbol{u}^a \in \Omega} \frac{|D_a^{'}(\boldsymbol{u}^a) - D_a(\boldsymbol{u}^a)|}{D_a^{'}(\boldsymbol{u}^a) + D_a(\boldsymbol{u}^a)}. \tag{3.11}$$



(a) *Boundary points get warped to pixels outside image.*

(b) *Occlusions render different depths for corresponding pixels.*

(c) *One camera may see further than the other.*

Figure 3.11: *Different cases that give invalid geometric errors between two views.*

Masking co-visible points

The loss in (3.11) assumes all pixels in one image have corresponding pixels in the other, which is not true for a moving camera (or world). Several cases invalidate this assumption, and they must be handled accordingly when calculating the geometric error. The most common approach is to mask out not co-visible points, leaving only the pixels that view the same 3D point between multiple images. The geometric error in (3.10) will then only consider pixels $\boldsymbol{u}^b$ that have corresponding pixels $\boldsymbol{u}^a$ which also view the same 3D point.

Figure 3.11 displays the different cases where the geometric error fails between two views. In Figure 3.11a, one camera sees points that are outside the view of the other, and these boundary pixels get warped to pixel coordinates outside the image plane. Occlusions, as shown in Figure 3.11b, are perhaps the most challenging to mask out and give different depths due to objects blocking the view in one image but not the other. The last case considered in this work is a limitation of the maximum detectable depth in the cameras, where one camera may see further than the other, shown in Figure 3.11c.

Another important case is that of moving objects. The world is rarely static, and most scenes will have dynamic elements that also invalidate the geometric error. The dataset used in this work, however, has completely static scenes, and this case will therefore not be considered here.

There are multiple ways to address the cases in Figure 3.11, and they are greatly simplified when the relative pose $\mathbf{T}_{ab}$ and depths D_a and D_b are known exact, which will be assumed here. Algorithm 1 summarizes the method used to calculate the geometric error with masking, where D_a^{gt} and D_b^{gt} are the ground truth depth maps and D_b is the predicted depth map. The algorithm iterates over every pixel in the image pair, skips pixels that fall under masking conditions and returns the average geometric error for the image pair. Boundary pixels are pixels that get warped to coordinates outside the image dimensions, and occluded pixels correspond to ground truth 3D points that, when transformed into the other view, have a different depth from the other view. Pixels outside of the depth range correspond to points that exceed the maximum depth when transformed. Note that these masking conditions rely on the ground truth depths of both views, in addition to the relative pose between the views, and the calculation is therefore only possible for supervised methods.

Because the procedure calculates the error iteratively over the pixels and not all at once, the computational overhead is quite significant for training. A subset of pixels is instead randomly

Algorithm 1 Algorithm for calculating the geometric error with masking. Mappings to and from homogeneous and Cartesian coordinates are skipped for notational simplicity, but are present for every transformation, projection and backprojection.

procedure GEOMETRICERROR($\mathbf{T}_{ab}$, D_a^{gt}, D_b^{gt}, D_b)
  $\mathbf{U}^b \leftarrow$ GeneratePixels(h, w) ▷ Pixel coordinates for entire image
  $\mathbf{X}^b \leftarrow \pi^{-1}(\mathbf{U}^b, D_b)$
  $\mathbf{X}^a \leftarrow \mathbf{T}_{ab}\mathbf{X}^b$
  $D_a^{'} \leftarrow |\mathbf{X}^a|_z$
  $\mathbf{X}_{gt}^b \leftarrow \pi^{-1}(\mathbf{U}^b, D_b^{gt})$
  $\mathbf{X}_{gt}^a \leftarrow \mathbf{T}_{ab}\mathbf{X}_{gt}^b$
  $D_a^{'gt} \leftarrow |\mathbf{X}_{gt}^a|_z$
  $\mathbf{U}^a \leftarrow w(\mathbf{U}^b, D_b, \mathbf{T}_{ab})$
  $e_g \leftarrow 0$
  $n \leftarrow 0$
  for $\boldsymbol{u}^a, \boldsymbol{u}^b$ in $\mathbf{U}^a, \mathbf{U}^b$ **do**
    if $\boldsymbol{u}^a < (0, 0)$ or $\boldsymbol{u}^a > (h, w)$ **then** ▷ Boundary masking
      continue
    $d_a^{'gt} \leftarrow D_a^{'gt}(\boldsymbol{u}^b)$
    if $d_a^{'gt} > d_{\max}$ **then** ▷ Maximum depth masking
      continue
    $d_a \leftarrow$ BilinearInterpolation$(\boldsymbol{u}^a, D_a^{gt})$
    occlusion error $\leftarrow |d_a - d_a^{'gt}|$
    if occlusion error > occlusion threshold **then** ▷ Occlusion masking
      continue
    $d_a^{'} \leftarrow D_a^{'}(\boldsymbol{u}^b)$
    $e_g \leftarrow e_g + |d_a - d_a^{'}|$
    $n \leftarrow n + 1$
  return e_g/n

sampled for each image pair to reduce the added training time. The number of sampled pixels should be low enough to allow for feasible training times but simultaneously high enough to give a sufficient signal.

3.3.4 Depth refinement network

The idea of multi-view consistency is to force the network to use information from all images. A possible problem is a long path from the images to the final prediction. As the multi-view consistency is implemented as a loss function at the very end, it may not influence the first layers sufficiently. Additionally, as it imposes consistency in *depth* and not any photometric measure, it may be more useful on feature maps that relate to depth.

Instead of using multi-view consistency on the Depth-CVAE, and additional *depth refinement network*, RefineNet, is added to produce consistent depth maps from multiple views. The network, as seen with the baseline Depth-CVAE in Figure 3.12, is an autoencoder that solely works on depth and takes three consecutive predictions from the single-view Depth-CVAE to produce refined estimates of the predictions. Additional details can be seen in Table D.1. At run-time, this can also work in a sliding-window manner with the single-view Depth-CVAE, where the two previous predictions are used together with the current prediction to produce a consistent current

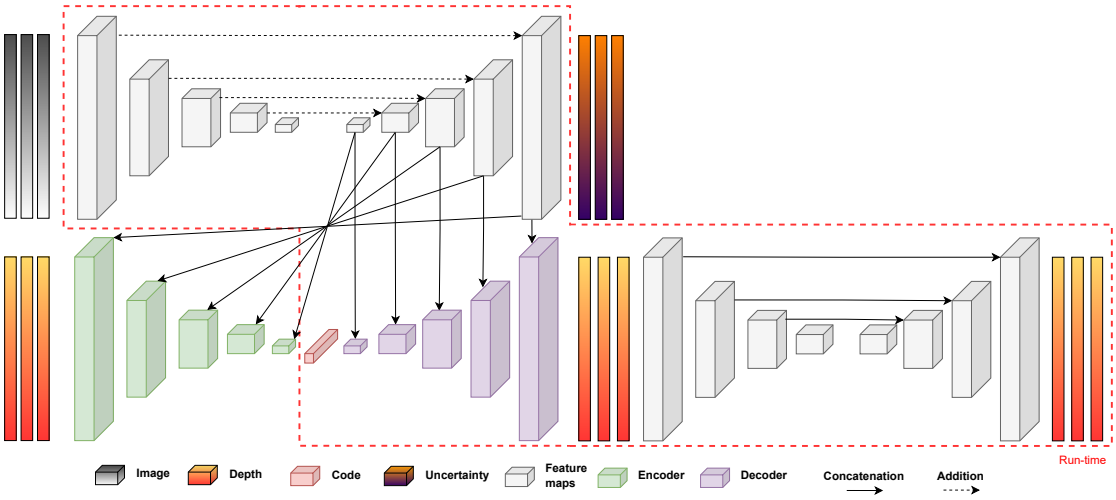


Figure 3.12: *Depth-CVAE with subsequent refinement by a separate depth refinement network.*

prediction. The added overhead is thus only RefineNet itself.

The Depth-CVAE is frozen during training to reduce overhead and is solely used as a means to provide realistic depth maps for refinement. RefineNet is then supervised with an ℓ_1 reconstruction loss and the multi-view geometric loss.

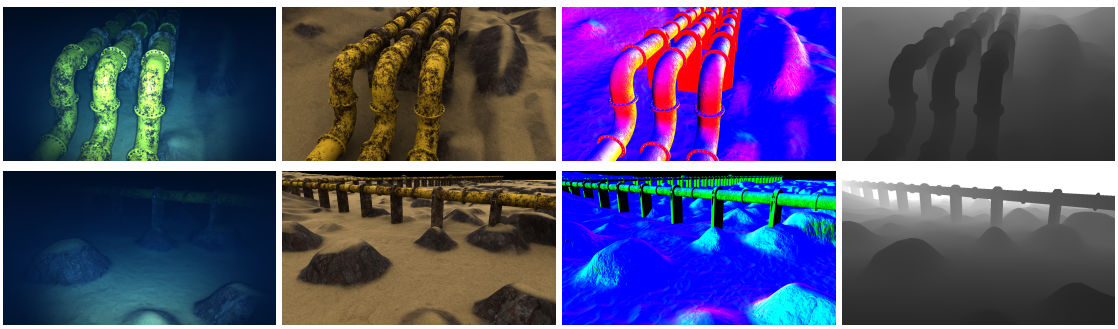
4 | Results and Discussion

In this chapter, the results and discussion for the experiments in Chapter 3 are presented both qualitatively and quantitatively. The VAROS Synthetic Underwater dataset is first presented, with some discussion around its characteristics for deep learning, before the specific experimental setup used in the experiments is described. Finally, the results from all experiments are shown, demonstrating the performance of the Depth-CVAE with each modification.

4.1 Dataset

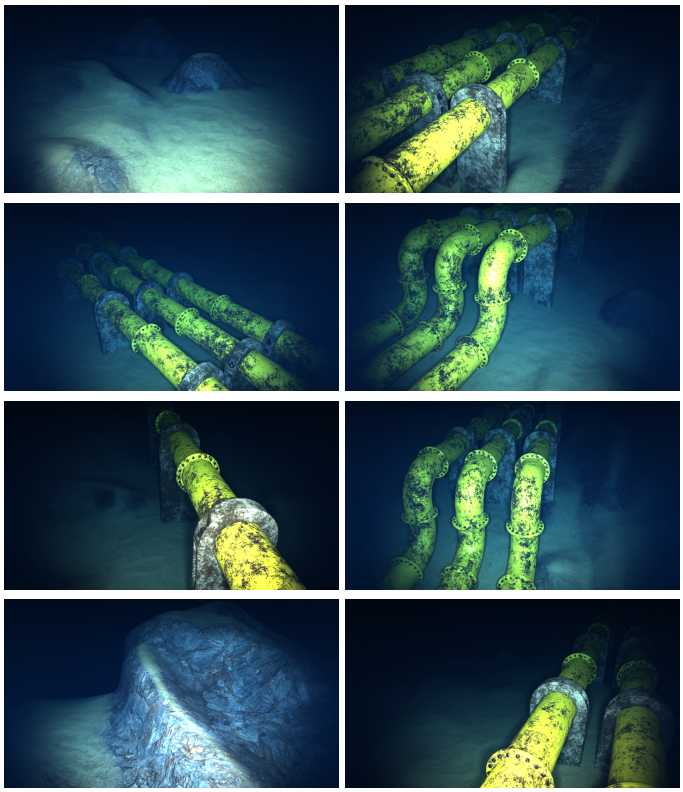
The VAROS Synthetic Underwater[41] dataset provides photorealistic synthetic RGB images generated in an artificial environment, with corresponding precise ground truth depth images. The benefit of using synthetic data is that the much-needed ground truth depth is available. However, the extent of realism in the synthetic scene is a bottleneck, mainly how well the photorealistic images translate to real scenes. This was shown in [1], where a model trained on VAROS was tested on actual underwater footage, and real scenes will therefore not be considered in this work.

Today, VAROS consists of 4715 rendered RGB images with corresponding depth maps, surface normal maps, and uniformly lighted images with no water. Additionally, the exact pose of each image relative to a static world frame is given. Two examples of the scene and its corresponding data can be seen in Figure 4.1. As VAROS provides a realistic underwater environment, modeling the varying illumination and visual degradation present in underwater scenes, it serves as a good test bench for real underwater images. At the current time, however, the dataset contains far too few images for training deep models, especially considering that the targeted application is reliable and accurate predictions used in a maneuvering robot. The current sequence of data



(a) *Underwater RGB* ***(b)*** *Uniform lighting* ***(c)*** *Surface normals* ***(d)*** *Depth*

Figure 4.1: *Two examples (top and bottom row) from the VAROS dataset.*



(a) *Training split.* ***(b)*** *Test split.*

Figure 4.2: *Examples from the training set (left) and test set (right).*

is also sequential, taken from a trajectory of a supposed robot, meaning consecutive frames are highly similar. This reduces the number of unique scenes seen by the model even further and specializes it heavily on the sequence on which it is trained. Therefore, choosing a representative test set from the limited data is also a challenge.

Choosing the training and test splits

The test set for VAROS is challenging to choose due to its limited size and, most notably, the sequential recording of data. Random images can therefore not be removed and used in the test set, as they will have very similar images in the training set, giving a highly biased indication of the model's performance.

Instead, a large sequence of data was removed and used as the test set, such that the test set did not contain images that were consecutive or very similar to any image in the training set. More specifically, every image viewing underwater pipes from the *right* side was moved from the training set and into the test set. As seen in Figure 4.2, the training set contains images of underwater seafloor, landscape, and pipes viewed from the left, while the test set exclusively contains pipes viewed from the right. Thus, when evaluating the model on VAROS, it is primarily the reconstruction of *pipes* that is evaluated. The overall distribution of pipes vs. not pipes in the dataset is therefore essential, in addition to the distribution of pipes viewed from the left vs. right. Table 4.1 shows the number of images containing pipes and which view they are in. As seen from the data distribution, only 22.9% of the training set contains pipes, while the rest views the seabed and underwater landscape. This should, therefore, not bias the model too heavily towards pipes, and the risk of overfitting is reduced to some extent. However, the angle

Images	Count	Pipes	Left	Right
0 - 225	226	✗	-	-
226 - 900	675	✓	✓	✗
901 - 1355	455	✓	✗	✓
1356 - 2700	1345	✗	-	-
2701 - 3000	300	✓	✓	✗
3001 - 4714	1714	✗	-	-

Table 4.1: *Distribution of images containing pipes, either viewed from the left or right.*

of view is relatively unchanged as the simulated vehicle is driving forward at all times. This biases the model towards the same views, and samples containing a pipe from e.g. the side or directly above, may give poor performance.

In Chapter 3.2.7, a note was made on data augmentation and why it is left out of this work. It should be stated that data augmentations fit very well on VAROS due to its small size, but if one were to use the common augmentations, any *horizontal flipping* should be avoided. This is because of the training and test splits, specifically that a horizontally flipped sample containing a pipe in the training set will be highly similar to the samples in the test set.

As the images are consecutive, samples at the start and end of the test set will be highly similar to their neighboring images in the training set. The neighboring images in the training set are, therefore, completely discarded to reduce the bias towards the training set further, such that there is a sufficient gap between the training set and the test set. Removing 50 images from the training set on both sides of the test set gives a sufficient gap, resulting in a final training set consisting of images 0-850 and 1405-4714 and a test set with images 901-1355. A dedicated validation set has not been made due to the small size of the dataset, and the model is therefore validated by the test set during training, giving a slight bias towards the test set when tuning the model. However, fine-tuning of the models has not been conducted in this work, so no efforts have been made to remove this bias.

4.2 Experiments

The single-view and multi-view Depth-CVAE were trained and tested on VAROS but without substantial hyperparameter-tuning or other forms of fine-tuning. The goal is not to result in one best model but to see what more considerable changes affect the performance of the models. Performance is most easily inspected by quantitative metrics, as changes in qualitative results are challenging to view, so most of the presented results will be in the form of performance metrics unless qualitative results give sufficient insight.

4.2.1 Experimental setup

All experiments were conducted on a NVIDIA GeForce GTX 1060 6GB with hyperparameters proposed in [11], using the Adam optimizer with learning rate $\alpha = 0.0001$ and $\beta_1 = 0.9$, $\beta_2 = 0.999$, training for 40 epochs using a small batch size of 4, due to the memory-constraints on the GPU. The weight on the KL divergence was gradually increased after eight epochs. The images were normalized using the calculated mean and standard deviation, and the depth values were transformed to the range [0, 1] by the proximity parameterization proposed in [11]. Both images and depth maps were resized to 288×512. The dimension of the code was set to 128.

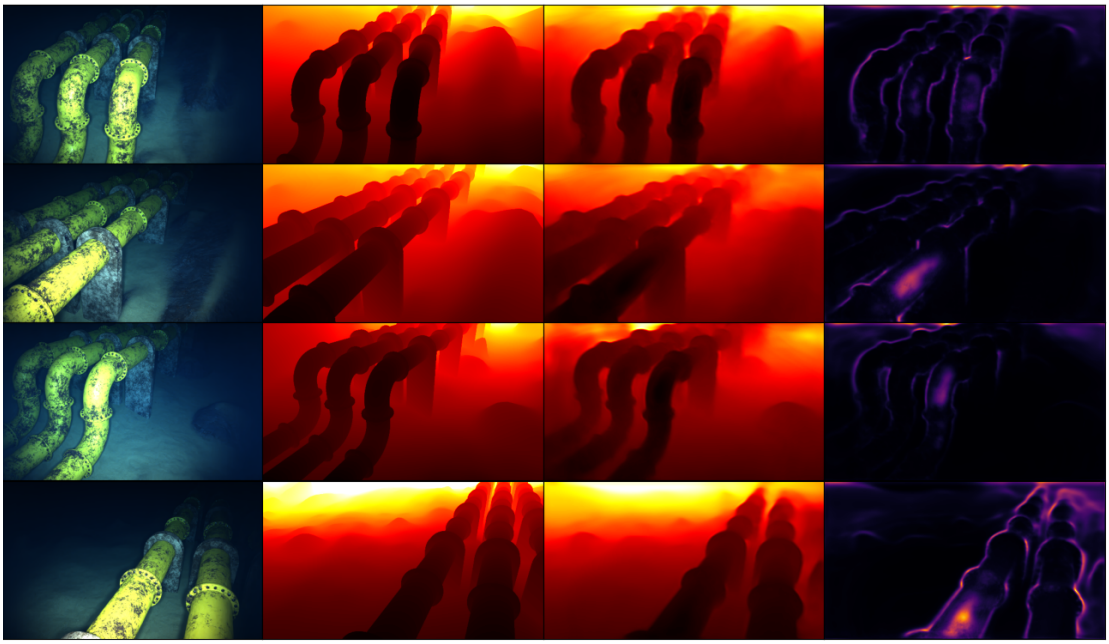


Figure 4.3: *Results from the baseline Depth-CVAE using RGB as input. From left to right: RGB image, ground truth depth, predicted depth and uncertainty of predicted depth. Brighter areas indicate higher values.*

	RMSE	logRMSE	ARD	SRD	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
RGB	**0.02202**	**0.00134**	**0.02611**	**0.00107**	**0.99098**	**0.99999**	**1.0**
Grayscale	0.03641	0.00578	0.04928	0.00267	0.98245	0.99968	**1.0**

Table 4.2: *Quantitative results from the baseline Depth-CVAE using RGB and grayscale as input. Lower values are better for the left side, while higher values are better for the right side. Bold values are comparatively better.*

4.2.2 Experimental results: Single-view

This section presents the results from the single-view Depth-CVAE and its modifications proposed in Chapter 3.2, with accompanying discussion for every experiment.

Baseline

Figure 4.3 and Figure 4.4 display qualitative results from four examples of the VAROS test set, using RGB and grayscale as input, respectively. The predicted depth maps in Figure 4.3 are backprojected to their point clouds in Figure 4.5. In Table 4.2, the models are compared using quantitative metrics calculated over the entire test set. Additional results can be seen in Appendix E.

By comparing Figure 4.3 and Figure 4.4, one can clearly see that using the full RGB information of the underwater images provides important details to the network, significantly outperforming luminance-only predictions. Especially fine-grained details are preserved when using RGB images, which appear more smoothed when using grayscale images. The qualitative difference can also be seen from the point clouds in Appendix E. The quantitative difference, as seen in Table 4.2, also shows the largest gap in metrics between two models compared to all other experiments.

Although this result may not seem too significant, it is important to note that grayscale images are commonly used instead of their RGB counterparts, and in many cases, they perform just as

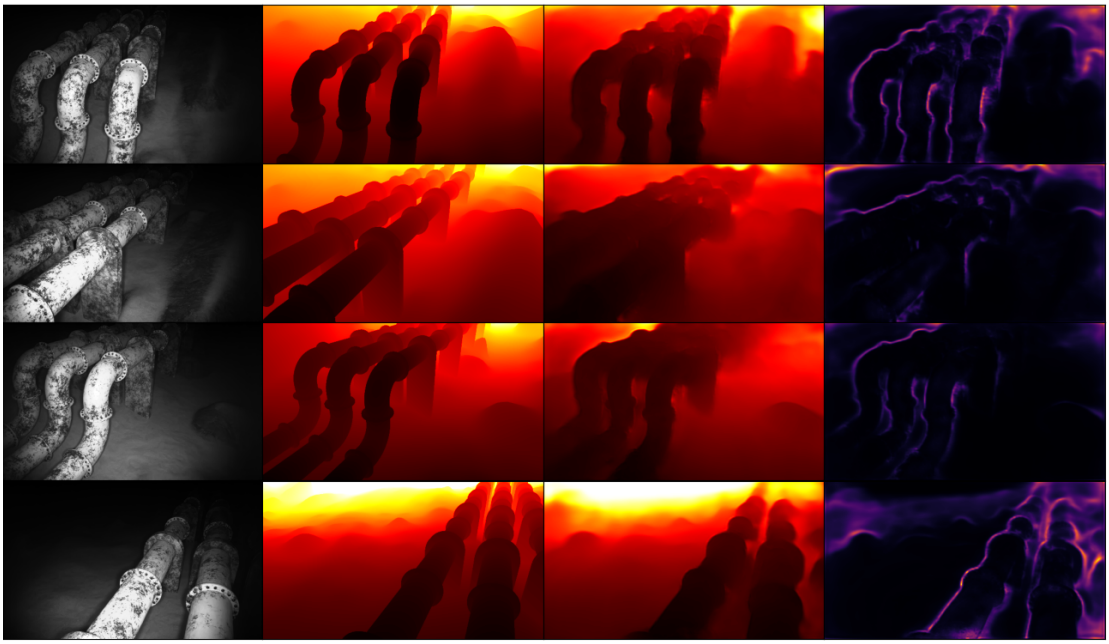


Figure 4.4: *Results from the baseline Depth-CVAE using grayscale as input. From left to right: Grayscale image, ground truth depth, predicted depth and uncertainty of predicted depth. Brighter areas indicate higher values.*

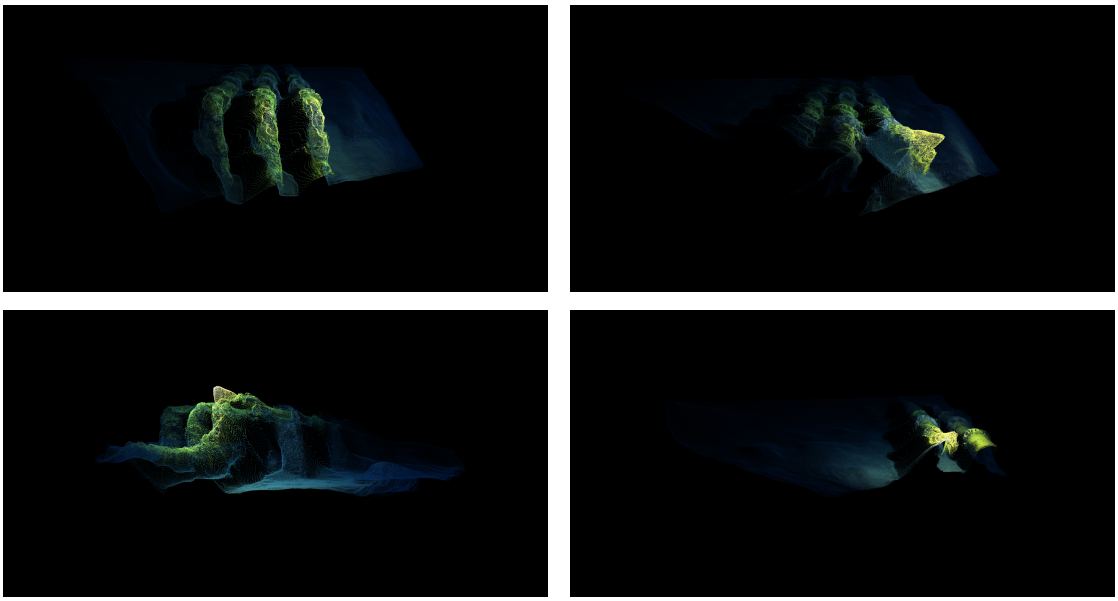


Figure 4.5: *Point clouds generated by the depth maps in Figure 4.3 from the baseline Depth-CVAE using RGB as input.*

	KITTI			NYU-Depth V2		
	ARD	SRD	$\delta < 1.25$	ARD	SRD	$\delta < 1.25$
BinsFormer[43]	0.052	0.151	0.974	0.094	-	0.925
DPT-Hybrid[17]	0.062	-	0.959	0.110	-	0.904
LapDepth[44]	0.059	-	0.962	0.105	-	0.895
Eigen *et al.*[8]	0.190	1.515	0.692	0.215	0.212	0.618

Table 4.3: *Comparison of relative metrics between current state-of-the-art deep learning monocular depth estimation methods on the KITTI Eigen split and NYU-Depth V2 datasets.*

well. As mentioned in Chapter 3, [11] observed no significant decrease in performance from grayscale with a similar model. The difference, however, is the data the model views, and it is clear that much information is lost for underwater images when converting to grayscale. All other experiments were therefore conducted with RGB as input to the model.

The estimated aleatoric uncertainty can also be viewed on the far right of Figure 4.3. The model allocates the highest uncertainty at edges, highly reflective surfaces, and points at a great distance from the camera. By inspecting the corresponding point clouds in Figure 4.5, the areas with the highest uncertainty are also the areas with the poorest depth predictions. This intuitively seems beneficial, as poor predictions can easily be discarded by their uncertainties. It is, however, difficult to say how true these uncertainties are to their probabilistic interpretations. For instance, in the luminance-only predictions of Figure 4.4, the allocated uncertainty for fine-grained regions is generally lower than when using RGB, even though the predictions are poorer (in the sense that they are smooth). This illustrates that the estimated uncertainty appears larger for regions with discontinuities in depth and not necessarily where the model allocates uncertainty in the images. On the other hand, another interpretation is that this model is falsely confident in these regions, so it is difficult to say exactly how good these uncertainty estimates are.

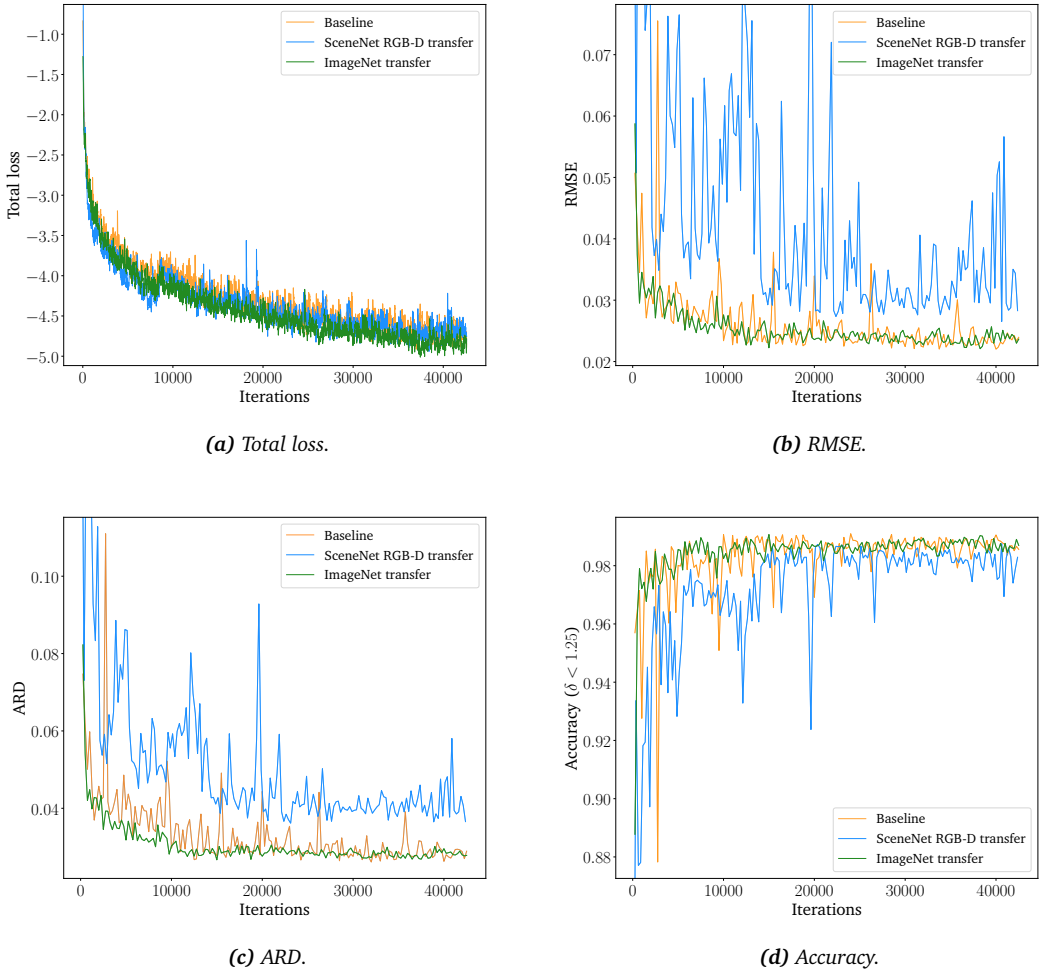
A note should also be made to the quantitative performance compared to the performance of other methods in the literature, with the consideration that the other methods are evaluated on different datasets. A comparison between some of the current state-of-the-art is shown in Table 4.3, comparing performance on the KITTI Eigen split[8] and NYU-Depth V2[42]. It should be stated that these datasets are much larger and more varied than VAROS, making them more challenging. As seen, the Depth-CVAE outperforms all these methods according to the relative performance metrics. This does not, however, mean that it is better than any of them, and more conclusive arguments should be made after evaluating the Depth-CVAE on the same datasets. The point here is that it performs surprisingly well on VAROS according to the quantitative metrics compared to what the state-of-the-art achieves on other data. This further highlights the inadequate properties of VAROS for evaluating deep learning methods, specifically that the small size and variation in the scenes make it easy to learn.

The baseline will be referred to as the baseline with RGB input from here on, as the baseline with grayscale images proved inferior compared to RGB.

Transfer learning

While the weights for ImageNet were pre-computed, the weights for SceneNet RGB-D were transferred from a Depth-CVAE trained on SceneNet RGB-D for 15 epochs using a batch size of 16. Only the training splits 0, 1, 2, 3, 4 and 5 from the official training set[1] were used for training, resulting in a total of 1.8M images, while training split 16 was used for validation.

[1] Downloaded from `https://robotvault.bitbucket.io/scenenet-rgbd.html`



(a) *Total loss.*

(b) *RMSE.*

(c) *ARD.*

(d) *Accuracy.*

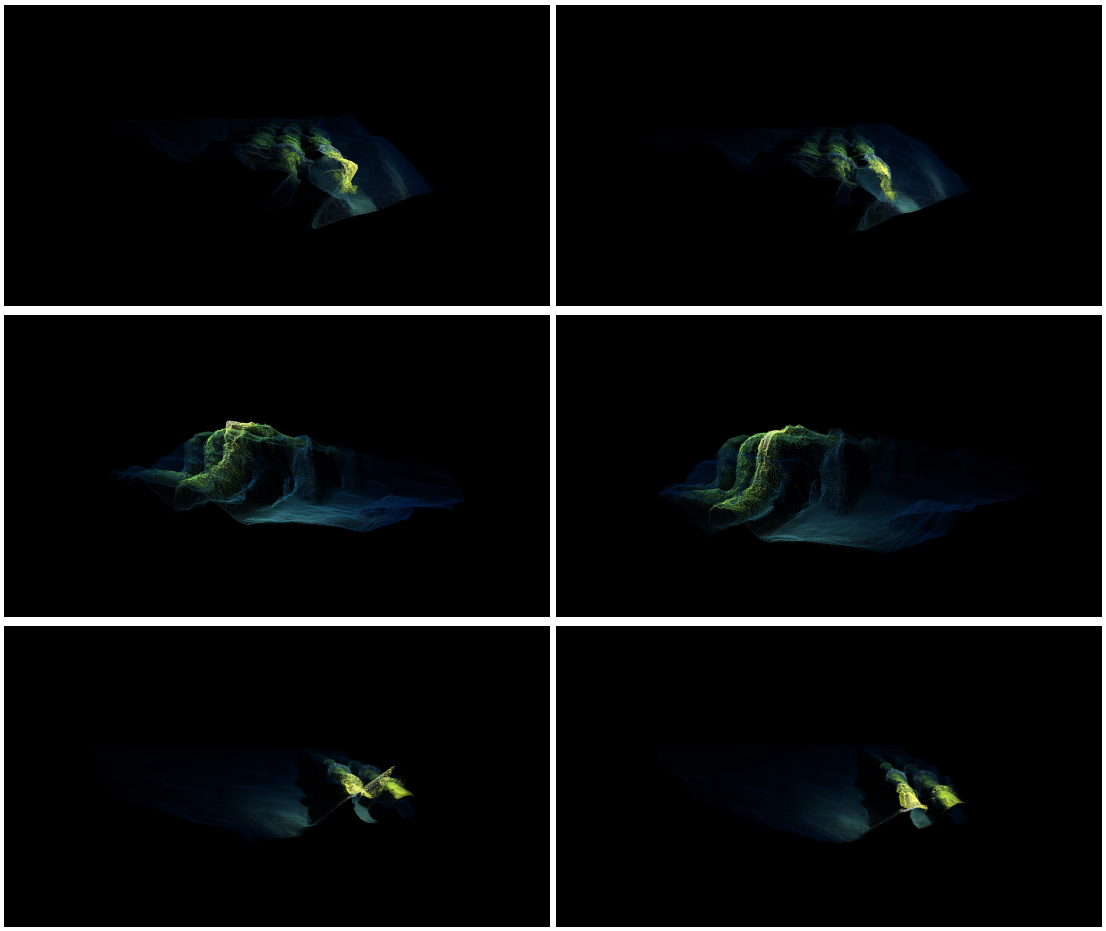
Figure 4.6: *Comparison of the convergence and performance using the baseline, SceneNet RGB-D transfer learning and ImageNet transfer learning.*

The KL annealing milestone was set to epoch 5, and the learning rate was decreased by a factor of 0.1 at epochs 10 and 13. Training parameters were otherwise identical to Chapter 4.2.1. Figure 4.6 displays the convergence properties and overall performance during training when using ImageNet or SceneNet RGB-D transfer learning, while a quantitative comparison of the models can be seen in Table 4.4.

From the results of using transfer learning in Figure 4.6, it can be seen that pre-trained ImageNet weights improve convergence and stability during training. This is not too surprising, as this is a relatively standard way of improving training, but it shows that ImageNet weights are comparative to those learned from VAROS. SceneNet pre-trained weights did not ease training and gave overall worse performance compared to the baseline. This is likely because SceneNet

	RMSE	logRMSE	ARD	SRD	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Baseline	**0.02202**	0.00134	**0.02611**	**0.00107**	**0.99098**	**0.99999**	**1.0**
ImageNet	0.02210	**0.00120**	0.02667	0.00108	0.99070	0.99983	**1.0**
SceneNet	0.02654	0.00200	0.03615	0.00155	0.98754	0.99969	**1.0**

Table 4.4: *Quantitative results using ImageNet and SceneNet RGB-D transfer learning. Lower values are better for the left side, while higher values are better for the right side. Bold values are comparatively better.*



(a) *Edge-aware smoothness.* ***(b)*** *Uncertainty-aware smoothness.*

Figure 4.7: *Comparison between using edge-aware smoothness and the proposed uncertainty-aware smoothness as a geometric prior.*

contains very different scenes from VAROS, and the learned weights relating to geometry will therefore not be transferable. Additionally, since the baseline network is trained for 40 epochs, it is safe to assume it learns the structure of VAROS sufficiently. Any transferred weights from other datasets may, therefore, not improve the performance. However, an interesting experiment could be to test the model on an underwater structure it has not seen, where SceneNet pre-trained weights may prove useful.

Uncertainty-aware smoothness

The penalty on the edge-aware smoothness in (3.4) was set to $\lambda = 1$ and activated from the start of training, while the uncertainty-aware smoothness was activated after 20 epochs to ensure converged and consistent uncertainty predictions. In Figure 4.7, a comparison is made between the edge-aware smoothness and uncertainty-aware smoothness for predicting piece-wise smooth geometry (except at edges). The results can be compared to the baseline in Figure 4.5 as well. In Table 4.5, the quantitative comparison between the models can be seen.

In the point clouds of the baseline model, there are several cases of outliers, specifically in areas with substantial reflections. The idea behind the smoothness priors (3.4) and (3.5) is to remove these outliers by forcing the local geometry to be smooth. The edge-aware smooth-

	RMSE	logRMSE	ARD	SRD	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Baseline	0.02202	0.00134	0.02611	0.00107	0.99098	**0.99999**	**1.0**
EA	**0.02122**	0.00134	**0.02576**	**0.00100**	**0.99162**	0.99993	**1.0**
UA	0.02253	**0.00116**	0.02653	0.00112	0.98938	0.99994	**1.0**

Table 4.5: *Quantitative results using edge-aware (EA) and uncertainty-aware (UA) smoothness. Lower values are better for the left side, while higher values are better for the right side. Bold values are comparatively better.*

	RMSE	logRMSE	ARD	SRD	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Baseline	0.02202	**0.00134**	0.02611	0.00107	0.99098	**0.99999**	**1.0**
MS-D	0.02436	0.00156	0.02836	0.00124	0.98956	0.99958	**1.0**
MS-S	**0.02167**	0.00147	**0.02571**	**0.00103**	**0.99203**	0.99992	**1.0**

Table 4.6: *Quantitative results using multi-scale loss with decoupled predictions (MS-D) and multi-scale loss using predictions in subsequent layers (MS-S). Lower values are better for the left side, while higher values are better for the right side. Bold values are comparatively better.*

ness limits the outliers to some extent but struggles to remove them completely. This is argued by considering that image gradients are poor approximations of edges and detect edges at the texture-rich smooth surface of the pipes. However, the uncertainty-aware smoothness reduces these outliers completely and makes the model predict geometry that is coherent with what one should expect from the images.

A significant result is made from the smoothness experiments: the edge-aware smoothness performs quantitatively better than the uncertainty-aware smoothness, even though the qualitative results are better for the uncertainty-aware smoothness. As seen in Table 4.5, the edge-aware smoothness outperforms the other models on almost all metrics. This shows that quantitative metrics alone may be a poor way to evaluate a model, and a decision must be made on what criteria should be decisive for the specific task. Additionally, since the quantitative metrics are calculated over the entire depth maps, the areas at the boundaries are equally weighted as the pipes. According to the overall metrics, a model with worse performance might perform better in reconstructing the pipes, but this will not be revealed by quantitative metrics alone.

Multi-scale loss

Table 4.6 displays the quantitative comparison between the models using multi-scale loss, either with multi-scale predictions decoupled from the rest of the network or with multi-scale predictions concatenated with subsequent layers.

Using multi-scale loss with predictions in subsequent layers proved superior to decoupled predictions, but it also requires more parameters, which may explain the performance boost. However, multi-scale loss with decoupled predictions gave a worse performance than the baseline, and as this is the most common way to implement multi-scale loss, it should perhaps be revised.

Depth in the top stream

In Table 4.7, the quantitative results of using depth in the top stream and depth with uncertainty in the bottom stream of the Depth-CVAE is shown.

The experiment of predicting depth in both the top and bottom stream, with uncertainty in the bottom stream, produces much better results than the baseline, relative to the difference between the baseline and other models. This confirms the hypothesis that the CVAE is better conditioned when the top stream predicts depth, unlike uncertainty which is commonly predicted

	RMSE	logRMSE	ARD	SRD	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Baseline	0.02202	0.00134	0.02611	0.00107	0.99098	**0.99999**	**1.0**
TopDepth	**0.02141**	**0.00124**	**0.02591**	**0.00103**	**0.99357**	0.99990	**1.0**

Table 4.7: *Quantitative results for predicting depth in the top stream and depth with uncertainty in the bottom stream of the Depth-CVAE. Lower values are better for the left side, while higher values are better for the right side. Bold values are comparatively better.*

	RMSE	logRMSE	ARD	SRD	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Baseline	0.02202	**0.00134**	0.02611	0.00107	0.99098	**0.99999**	**1.0**
Sparse-Ma	0.02284	0.00171	0.02730	0.00116	0.99052	0.99998	**1.0**
Sparse-MS	**0.02163**	0.00143	**0.02591**	**0.00102**	**0.99251**	0.99988	**1.0**

Table 4.8: *Quantitative results using auxiliary sparse depth at input (Sparse-Ma) and auxiliary sparse depth at multiple scales (Sparse-MS). Lower values are better for the left side, while higher values are better for the right side. Bold values are comparatively better.*

in the top stream[11], [19], [20].

Auxiliary sparse depth

Sparse depth is sampled at 1000 detected ORB features for each depth map. The results from the test set can be seen in Table 4.8, comparing the method similar to Ma *et al.*[38] and the proposed multi-scale depth.

The multi-scale sparse depth is seen to produce better results than the commonly used strategy in the literature, which does not improve on the baseline. Although Ma *et al.*[38] is the most renowned work for using auxiliary sparse depth in depth completion, it has been improved upon in later works, e.g. [23], so the proposed method may not be state-of-the-art as an auxiliary method.

4.2.3 Experimental results: Multi-view

This section presents the results and discussion for the multi-view Depth-CVAE and its modifications proposed in Chapter 3.3.

Baseline

The qualitative performance of the baseline multi-view Depth-CVAE is shown in Figure 4.8, where the results from the most current of the three concatenated images are used. The point clouds generated from each depth map are displayed in Figure 4.9, and the quantitative performance on the test set can be seen in Table 4.9.

A detail worth repeating before going forward is that all multi-view methods use grayscale images as input due to the computational demand of using multiple RGB images exceeding the constraints of the experimental platform. As discussed in the single-view case, this heavily affects the model's performance on underwater data. The results from the multi-view models will therefore reflect this.

	RMSE	logRMSE	ARD	SRD	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Baseline	0.03642	0.00535	0.04809	0.00256	0.98391	0.99982	1.0

Table 4.9: *Quantitative results for the baseline multi-view Depth-CVAE. Lower values are better for the left side, while higher values are better for the right side.*

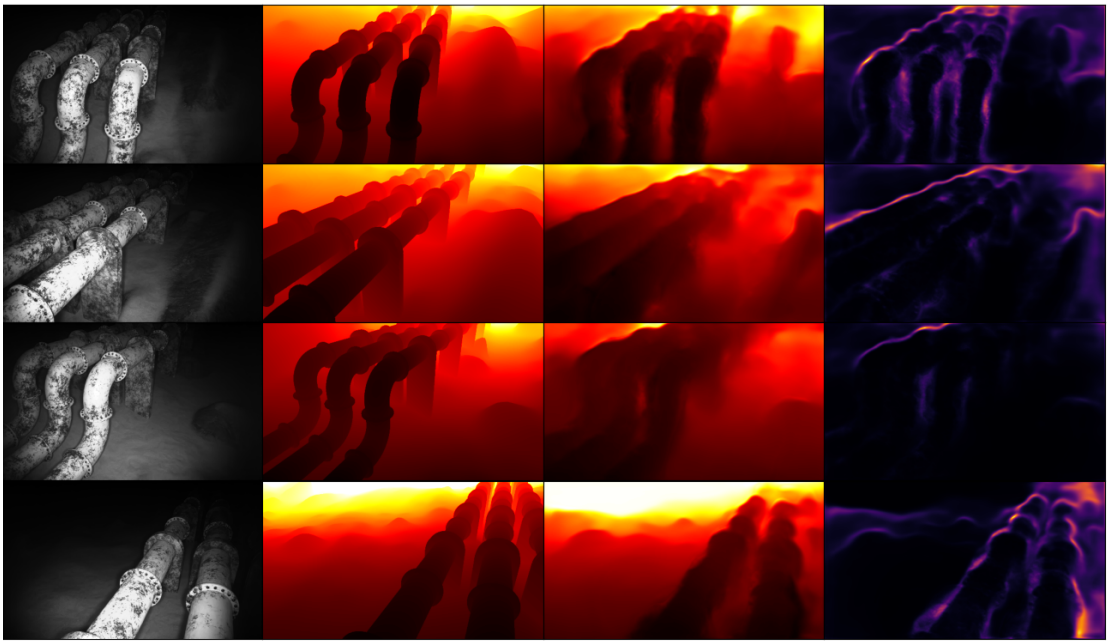


Figure 4.8: *Results from the baseline multi-view Depth-CVAE. From left to right: Grayscale image, ground truth depth, predicted depth and uncertainty of predicted depth. Brighter areas indicate higher values.*

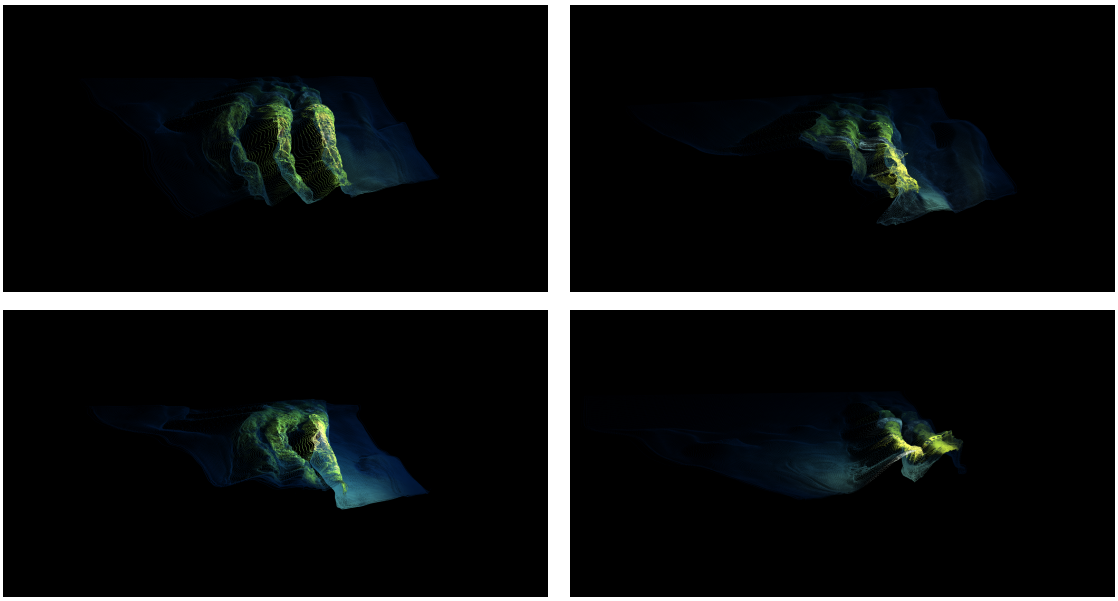


Figure 4.9: *Point clouds generated by the depth maps in Figure 4.8 from the baseline multi-view Depth-CVAE.*

	RMSE	logRMSE	ARD	SRD	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Baseline	**0.03642**	**0.00535**	**0.04809**	**0.00256**	0.98391	0.99982	**1.0**
MF-transfer	0.03841	0.00537	0.05052	0.00282	**0.98453**	**0.99988**	**1.0**

Table 4.10: *Quantitative results for the motion filter pre-trained multi-view Depth-CVAE. Lower values are better for the left side, while higher values are better for the right side. Bold values are comparatively better.*

Comparing the baseline of the multi-view model, Table 4.9, to the baseline of the single-view model with grayscale in Table 4.2, the results are slightly better for the multi-view model. However, this marginal increase in performance is not enough to justify that multi-view performs better than single-view. In model-based multiple view geometry, corresponding points between images are used to triangulate the depth of a pixel. The idea behind the multi-view network is to exploit this by providing multiple images and letting the network learn dependencies between feature maps to improve the estimates. This does not appear to be the case, and although it is difficult to reason on *why* for machine learning models, it is not unreasonable to conclude that the network has insufficient *guidance* during training to utilize all information. This could be solved by introducing more model-based constraints to the problem, which forces the network to fuse information from all inputs.

Motion filter pre-training

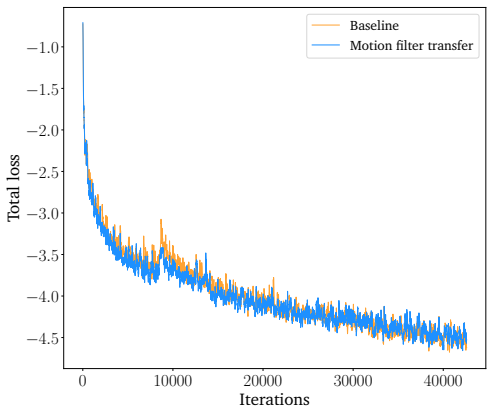
The motion autoencoder was trained for 80 epochs using a batch size of 8 on the KITTI raw dataset[39], and the original image resolution of 375×1242 was used throughout the training. The learning rate was set to 0.001 for the Adam optimizer and decreased by a factor of 0.1 after epochs 20 and 40. Although 80 epochs is a vast number of times to run over the dataset, with guaranteed exposure to overfitting for the task of predicting the next consecutive image, only the encoder of the motion autoencoder was transferred to the multi-view Depth-CVAE. As motion filters are general (to a large degree) and not specific to one dataset, the problem of overfitting is not relevant for this case.

From Figure 4.10, it can be seen that motion filter pre-training dramatically increases the rate of learning, starting at lower errors and higher accuracies, while also being more stable. The best performing models, however, come from the baseline, as seen in Table 4.10, except for accuracy metrics. Nonetheless, the main result is that motion filters are present in depth estimation networks using multiple views and that they can be learned in a self-supervised manner from video footage. A possibility could be that *any* pre-trained network on image data gives a better starting point than randomly initialized weights. To check this, an autoencoder trained on reconstructing the exact same images it sees was made. This did not give improved performance after transfer learning to the Depth-CVAE, so the proposed motion autoencoder actually learns motion filters.

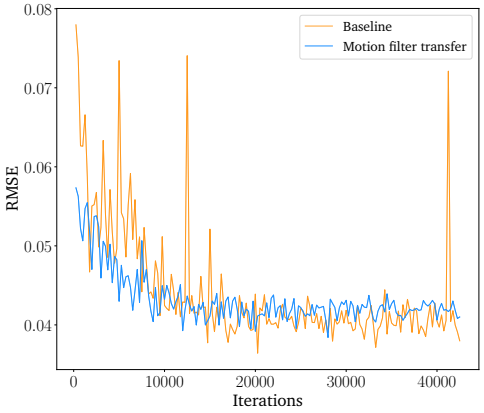
Multi-view consistency

The geometric error was calculated by sampling 200 pixel locations for each depth map. Consistency was only calculated between the target image and the previous images, not between the previous images themselves. Due to the computational cost of the geometric error, the number of epochs was reduced to 20. Results are shown in Table 4.11.

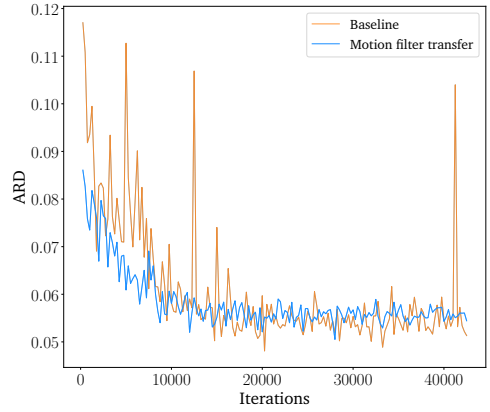
From the results of using multi-view consistency in the loss function, it appears the performance is degraded substantially compared to the baseline, although it should improve the model in theory. It is difficult to pinpoint the cause of the degradation, and an incorrect implementa-



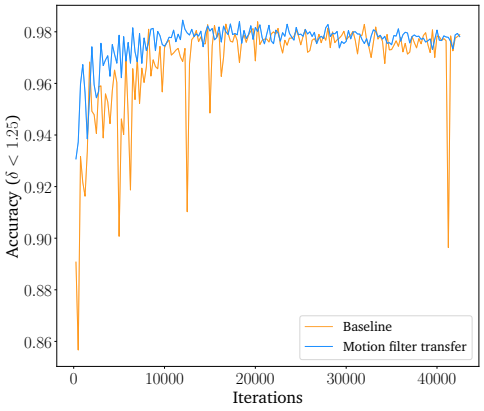
(a) Total loss.



(b) RMSE.



(c) ARD.



(d) Accuracy.

Figure 4.10: *Comparison of the convergence and performance between the baseline and motion filter pre-trained baseline.*

	RMSE	logRMSE	ARD	SRD	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Baseline	**0.03642**	0.00535	**0.04809**	**0.00256**	**0.98391**	**0.99982**	**1.0**
Consistency	0.04285	**0.00456**	0.05364	0.00336	0.97553	0.99961	**1.0**

Table 4.11: *Quantitative results when using multi-view consistency. Lower values are better for the left side, while higher values are better for the right side. Bold values are comparatively better.*

	RMSE	logRMSE	ARD	SRD	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Baseline	**0.03642**	0.00535	**0.04809**	**0.00256**	**0.98391**	**0.99982**	**1.0**
RefineNet	0.04059	**0.00482**	0.05586	0.00328	0.97287	0.99919	0.99999

Table 4.12: *Quantitative results for RefineNet. Lower values are better for the left side, while higher values are better for the right side. Bold values are comparatively better.*

tion may be the cause. However, the apparent problem is that too few pixels are sampled and calculated for each depth map. A total of 147456 pixel locations exist in each depth map, but only 200 are sampled at each training step due to the computational cost of calculating the geometric error. This is far from sufficient and may not give a significant enough signal for training. The pixels are also sampled uniformly, meaning there is no guarantee that important structure is consistent from one frame to the next, as pixels in the boundary regions may be sampled.

Depth refinement network

Same as for the multi-view consistency training, the depth refinement network was trained for 20 epochs with 200 pixel locations sampled for each depth map. Quantitative results can be seen in Table 4.12.

RefineNet was made under the hypothesis that multi-view consistency is poorly suited for a network that operates on both depth and images, but performs even worse. However, a problem here is that refining the depth maps solely by a reconstruction loss also degraded the depth maps, so the geometric error is not at fault by itself here.

4.2.4 Summary

A summary of the quantitative performance for all proposed methods is given in Table 4.13, additionally displaying which methods perform best according to the respective metrics. In order to compare the overall performance between the methods, the APS is calculated, shown at the far right. Note that the score for the multi-view methods is calculated by comparing to the best performing models from the single-view methods, such that the performance of the multi-view methods can be compared to the single-view methods.

The overall best performing model is the network predicting depth in the top stream, followed by the edge-aware smoothness and ImageNet transfer learning. These are, however, purely quantitative measures of performance, and qualitative performance is equally important, as argued for the uncertainty-aware smoothness. The APS also shows that many of the proposed modifications improve on their counterparts. Using depth in the top stream improves on the baseline, which uses uncertainty in the top stream, and multi-scale late fusion sparse depth performs better than single-scale early fusion sparse depth. Using edge-aware smoothness performs better than uncertainty-aware smoothness quantitatively, but not qualitatively, so a choice must be made for what is decisive. The multi-view methods are seen to be inferior to the single-view methods, but much is likely due to using grayscale images. However, every modification leads to worse performance, leaving much further work in this area.

Although an investigation into real-time performance is outside the scope of this work, it should be mentioned as it is highly relevant to the problem at hand. A robot mapping its surroundings at a low rate will struggle with localization and collision avoidance tasks. However, the exact rate of predictions will be hardware-dependent and challenging to assess on a single platform. All single-view methods have close to 46.4M parameters during inference, while the

	RMSE	logRMSE	ARD	SRD	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$	APS
Single-view								
Baseline	0.02202	0.00134	0.02611	0.00107	0.99098	**0.99999**	**1.0**	1.04015
ImageNet	0.02210	0.00120	0.02667	0.00108	0.99070	0.99983	**1.0**	1.02805
SceneNet	0.02654	0.00200	0.03615	0.00155	0.98754	0.99969	**1.0**	1.27676
EA	**0.02122**	0.00134	0.02576	**0.00100**	0.99162	0.99993	**1.0**	1.02273
UA	0.02253	**0.00116**	0.02653	0.00112	0.98938	0.99994	**1.0**	1.03113
MS-D	0.02436	0.00156	0.02836	0.00124	0.98956	0.99958	**1.0**	1.12005
MS-S	0.02167	0.00147	**0.02571**	0.00103	0.99203	0.99992	**1.0**	1.04572
TopDepth	0.02141	0.00124	0.02591	0.00103	**0.99357**	0.99990	**1.0**	**1.01654**
Sparse-Ma	0.02284	0.00171	0.02730	0.00116	0.99052	0.99998	**1.0**	1.11077
Sparse-MS	0.02163	0.00143	0.02591	0.00102	0.99251	0.99988	**1.0**	1.04015
Multi-view								
Baseline	**0.03642**	0.00535	**0.04809**	**0.00256**	0.98391	0.99982	**1.0**	**1.96698**
MF-transfer	0.03841	0.00537	0.05052	0.00282	**0.98453**	**0.99988**	**1.0**	2.03338
Consistency	0.04285	**0.00456**	0.05364	0.00336	0.97553	0.99961	**1.0**	2.05937
RefineNet	0.04059	0.00482	0.05586	0.00328	0.97287	0.99919	**1.0**	2.07754

Table 4.13: *Quantitative summary for all models. Lower values are better for the left and right side, while higher values are better for the middle. Bold values are comparatively better.*

multi-view methods have 45.5M, except for RefineNet, which with the Depth-CVAE has a total of 59.4M parameters. Unfortunately, many of the current state-of-the-art methods do not provide the size of their networks, so it is difficult to quickly assess how large the Depth-CVAE is compared to other methods.

5 | Conclusion

This thesis presented methods for improving supervised deep monocular depth estimation using convolutional neural networks. Despite being evaluated on and targeted at underwater scenes, many of the proposed methods are general and may be applied to other domains and are flexible to different architectures.

Necessary background information and theory for the implemented variational autoencoder were presented, which served as a baseline for evaluating the performance of the proposed modifications. Methods tested on, but not limited to, single-view depth estimation were then displayed. Leveraging uncertainty estimates for geometric smoothing was shown to produce more coherent depth maps than the current state-of-the-art geometric prior. Adding auxiliary sparse depth in a multi-scale late fusion scheme improved the quantitative performance compared to typical depth completion schemes using sparse depth. Additionally, and more specific to variational autoencoders, it was shown that predicting depth in the top stream and depth with uncertainty in the bottom stream improved results, in contrast to typical implementations.

Spatiotemporal networks using multiple views were then considered, with the hypothesis that multiple views would improve performance as for traditional model-based methods. Results on underwater images using 2D spatiotemporal networks indicate that little is learned between the views. They perform equally or worse than their single-view counterparts, even when enforcing consistency between predictions from multiple views. A more extensive investigation into 2D spatiotemporal networks is, therefore, needed.

5.1 Further work

The work done in this thesis is just a small contribution to the extensive field of depth estimation, and there is much that can be improved upon or further investigated. Possible further work is naturally to examine model-specific considerations more closely, but improvements on the dataset and surrounding elements should be made. The greatest weakness of this work is perhaps that the model is not transferable to real underwater scenes, which, after all, is the target application. The following lists some potential work in these areas.

Further research should be done into the successful deployment of deep learning models in real underwater environments:

- Ground truth depth is difficult to obtain underwater, if at all possible, meaning supervised methods are poorly suited for these scenarios. Today, self-supervised methods use assumptions on photometric consistency to extrapolate depth, but these assumptions fail underwater. Bypassing this issue, by e.g. proposing other metrics that are robust to visual degradation, will enable self-supervised methods on underwater data as well.

- A limitation of VAROS is that it does not provide a sufficiently realistic underwater environment to directly transfer a learned model into real underwater scenes, as shown in [1]. The simulator should be further developed to make the scenes as realistic as possible, potentially making supervised models work in real scenes.

Other than the aforementioned issue, several other efforts can be made to improve VAROS for deep learning:

- The limited size of the dataset has already been mentioned as an issue for deep learning, and several different sequences should be made to increase the training and test sets.
- The variety of scenes is today very limited, and more man-made structures or other commonly observed structures in real scenes should be added.

The depth estimation network implementation and pipeline can be improved upon in several areas:

- The strength of the code in the CVAE is not fully realized in the Depth-CVAE, and is currently not used during optimization. The original idea of [11] was to use the reduced dimensionality of the code to ease optimization for VSLAM, and this can be further investigated for multi-view consistency during training.
- The loss and calculated metrics should only be calculated for regions of interest, and not the entire image. Specifically, they should be calculated for areas that are *visible*, and not boundary regions that are completely dark.
- The multi-scale late fusion scheme for auxiliary sparse depth uses zero imputation for missing values, which is suboptimal as zero actually is a depth value. Using the normalized convolutions of [23] with confidence propagation could improve the performance of this method.
- The multi-view data loader skips a fixed set of frames between returned images to ensure enough motion, but a keyframe-based approach should instead be used based on the actual motion between the images.
- A faster implementation of the geometric error for multi-view consistency should be found, as the proposed algorithm is too slow for sampling a larger part of the depth maps during training.
- 2D spatiotemporal networks may not be suited for multiple view depth estimation at all, but 3D convolutional neural networks using cost volumes have been shown to produce good results[27]. Further experimentation with these networks may be a better alternative to 2D networks.

The Depth-CVAE can be evaluated more thoroughly:

- The models are not comparable to the state-of-the-art within depth estimation as they are only evaluated on VAROS. To compare the actual performance of the Depth-CVAE, it should be evaluated on one of the popular datasets like KITTI, EuRoC or NYU-Depth.
- Each modification is in this work evaluated by itself, but a model containing several modifications together should be tested and compared against the best performing model.

Bibliography

[1] A. T. Fagerli, *Deep monocular depth estimation for underwater autonomous vehicles*, Specialization project, 2021.

[2] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. USA: Cambridge University Press, 2004, ISBN: 0521540518.

[3] T. V. Haavardsholm, *A handbook in visual slam*, 2021. [Online]. Available: https://github.com/tussedrotten/vslam-handbook.

[4] A. Krizhevsky, I. Sutskever and G. E. Hinton, 'Imagenet classification with deep convolutional neural networks,' in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[5] K. Simonyan and A. Zisserman, 'Very deep convolutional networks for large-scale image recognition,' 2015. arXiv: 1409.1556 [cs.CV].

[6] K. He, X. Zhang, S. Ren and J. Sun, 'Deep residual learning for image recognition,' 2015. arXiv: 1512.03385 [cs.CV].

[7] O. Ronneberger, P. Fischer and T. Brox, 'U-net: Convolutional networks for biomedical image segmentation,' 2015. arXiv: 1505.04597 [cs.CV].

[8] D. Eigen, C. Puhrsch and R. Fergus, 'Depth map prediction from a single image using a multi-scale deep network,' 2014. arXiv: 1406.2283 [cs.CV].

[9] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari and N. Navab, 'Deeper depth prediction with fully convolutional residual networks,' 2016. arXiv: 1606.00373 [cs.CV].

[10] D. P. Kingma and M. Welling, 'Auto-encoding variational bayes,' 2014. arXiv: 1312.6114 [stat.ML].

[11] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger and A. J. Davison, 'Codeslam - learning a compact, optimisable representation for dense visual slam,' 2019. arXiv: 1804.00874 [cs.CV].

[12] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby and O. Winther, 'Ladder variational autoencoders,' 2016. arXiv: 1602.02282 [stat.ML].

[13] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz and S. Bengio, 'Generating sentences from a continuous space,' 2016. arXiv: 1511.06349 [cs.LG].

[14] H. Fu, C. Li, X. Liu, J. Gao, A. Celikyilmaz and L. Carin, 'Cyclical annealing schedule: A simple approach to mitigating kl vanishing,' 2019. arXiv: 1903.10145 [cs.LG].

[15] L. Deng, 'The mnist database of handwritten digit images for machine learning research,' *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[16] K. Sohn, H. Lee and X. Yan, 'Learning structured output representation using deep conditional generative models,' in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/8d55a249e6baa5c06772297520da2051-Paper.pdf.

[17] R. Ranftl, A. Bochkovskiy and V. Koltun, 'Vision transformers for dense prediction,' 2021. arXiv: 2103.13413 [cs.CV].

[18] N. Yang, L. von Stumberg, R. Wang and D. Cremers, 'D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry,' 2020. arXiv: 2003.01060 [cs.CV].

[19] J. Czarnowski, T. Laidlow, R. Clark and A. J. Davison, 'Deepfactors: Real-time probabilistic dense monocular slam,' *IEEE Robotics and Automation Letters*, vol. 5, no. 2, 721–728, 2020, ISSN: 2377-3774. DOI: 10.1109/lra.2020.2965415. [Online]. Available: http://dx.doi.org/10.1109/LRA.2020.2965415.

[20] X. Zuo, N. Merrill, W. Li, Y. Liu, M. Pollefeys and G. Huang, 'Codevio: Visual-inertial odometry with learned optimizable dense depth,' 2021. arXiv: 2012.10133 [cs.CV].

[21] H. Matsuki, R. Scona, J. Czarnowski and A. J. Davison, 'Codemapping: Real-time dense mapping for sparse slam using compact scene representations,' 2021. arXiv: 2107.08994 [cs.CV].

[22] A. Kendall and Y. Gal, 'What uncertainties do we need in bayesian deep learning for computer vision?,' 2017. arXiv: 1703.04977 [cs.CV].

[23] A. Eldesokey, M. Felsberg and F. S. Khan, 'Propagating confidences through cnns for sparse data regression,' 2018. DOI: 10.48550/ARXIV.1805.11913. [Online]. Available: https://arxiv.org/abs/1805.11913.

[24] D. Wofk, F. Ma, T.-J. Yang, S. Karaman and V. Sze, 'Fastdepth: Fast monocular depth estimation on embedded systems,' 2019. arXiv: 1903.03273 [cs.CV].

[25] C. Godard, O. M. Aodha, M. Firman and G. Brostow, 'Digging into self-supervised monocular depth estimation,' 2019. arXiv: 1806.01260 [cs.CV].

[26] N. Yang, R. Wang, J. Stückler and D. Cremers, 'Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry,' 2018. arXiv: 1807.02570 [cs.CV].

[27] Y. Yao, Z. Luo, S. Li, T. Fang and L. Quan, 'Mvsnet: Depth inference for unstructured multi-view stereo,' 2018. DOI: 10.48550/ARXIV.1804.02505. [Online]. Available: https://arxiv.org/abs/1804.02505.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, 'Pytorch: An imperative style, high-performance deep learning library,' in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[29] C. Li, *High quality, fast, modular reference implementation of SSD in PyTorch*, https://github.com/lufficc/SSD, 2018.

[30] D. L. MacAdam, 'Projective transformations of i. c. i. color specifications,' *J. Opt. Soc. Am.*, vol. 27, no. 8, pp. 294–299, 1937. DOI: 10.1364/JOSA.27.000294. [Online]. Available: http://opg.optica.org/abstract.cfm?URI=josa-27-8-294.

[31] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, 'Imagenet large scale visual recognition challenge,' 2014. DOI: 10.48550/ARXIV.1409.0575. [Online]. Available: https://arxiv.org/abs/1409.0575.

[32] J. McCormac, A. Handa, S. Leutenegger and A. J.Davison, 'Scenenet rgb-d: Can 5m synthetic images beat generic imagenet pre-training on indoor segmentation?,' 2017.

[33] C. Godard, O. Mac Aodha and G. J. Brostow, 'Unsupervised monocular depth estimation with left-right consistency,' 2016. DOI: 10.48550/ARXIV.1609.03677. [Online]. Available: https://arxiv.org/abs/1609.03677.

[34] J. Canny, 'A computational approach to edge detection,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986. DOI: 10.1109/TPAMI.1986.4767851.

[35] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, 'Feature pyramid networks for object detection,' 2016. DOI: 10.48550/ARXIV.1612.03144. [Online]. Available: https://arxiv.org/abs/1612.03144.

[36] T. Zhou, M. Brown, N. Snavely and D. G. Lowe, 'Unsupervised learning of depth and ego-motion from video,' 2017. DOI: 10.48550/ARXIV.1612.03144. [Online]. Available: https://arxiv.org/abs/1704.07813.

[37] C. Campos, R. Elvira, J. J. Gómez, J. M. M. Montiel and J. D. Tardós, 'ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM,' *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

[38] F. Ma and S. Karaman, 'Sparse-to-dense: Depth prediction from sparse depth samples and a single image,' 2017. DOI: 10.48550/ARXIV.1709.07492. [Online]. Available: https://arxiv.org/abs/1709.07492.

[39] A. Geiger, P. Lenz, C. Stiller and R. Urtasun, 'Vision meets robotics: The kitti dataset,' *International Journal of Robotics Research (IJRR)*, 2013.

[40] J.-W. Bian, Z. Li, N. Wang, H. Zhan, C. Shen, M.-M. Cheng and I. Reid, 'Unsupervised scale-consistent depth and ego-motion learning from monocular video,' 2019. DOI: 10.48550/ARXIV.1908.10553. [Online]. Available: https://arxiv.org/abs/1908.10553.

[41] P. G. O. Zwilgmeyer, M. Yip, A. L. Teigen, R. Mester and A. Stahl, 'The varos synthetic underwater data set: Towards realistic multi-sensor underwater data with ground truth,' in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, 2021, pp. 3722–3730.

[42] P. K. Nathan Silberman Derek Hoiem and R. Fergus, 'Indoor segmentation and support inference from rgbd images,' in *ECCV*, 2012.

[43] Z. Li, X. Wang, X. Liu and J. Jiang, 'Binsformer: Revisiting adaptive bins for monocular depth estimation,' 2022. DOI: 10.48550/ARXIV.2204.00987. [Online]. Available: https://arxiv.org/abs/2204.00987.

[44] M. Song, S. Lim and W. Kim, 'Monocular depth estimation using laplacian pyramid-based depth residuals,' *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 11, pp. 4381–4393, 2021. DOI: 10.1109/TCSVT.2021.3049869.

[45] K. B. Petersen and M. S. Pedersen, 'The matrix cookbook,' 2012, Version 20121115. [Online]. Available: http://www2.compute.dtu.dk/pubdb/pubs/3274-full.html.

A | Proofs

A.1 Marginal log-likelihood for VAEs

The aim is to find the best approximation $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ to the intractable $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, which can be done by minimizing the KL divergence (2.18) between the two distributions:

$$\mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\middle\|p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) - \log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right].$$

Using Bayes' rule (2.15) on $\log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, the KL divergence takes the form

$$\begin{aligned}\mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\middle\|p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) - (\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{z}) - p_{\boldsymbol{\theta}}(\boldsymbol{x}))\right] \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) - \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) - \log p_{\boldsymbol{\theta}}(\boldsymbol{z})\right] + p_{\boldsymbol{\theta}}(\boldsymbol{x}) \qquad \text{(A.1)}\end{aligned}$$

where $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is taken out of the expectation as it is not a function of $\boldsymbol{z}$. Finally, (A.1) is solved for the marginal log-likelihood,

$$\begin{aligned}p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\middle\|p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[-\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{z})\right] \qquad \text{(A.2)} \\ &= \mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\middle\|p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}),\end{aligned}$$

where the expectation in (A.2) is the variational lower bound.

A.2 KL divergence between two Gaussians

For two multivariate Gaussians, $\mathcal{N}_1(\mu_1, \Sigma_1)$ and $\mathcal{N}_2(\mu_2, \Sigma_2)$, the KL divergence takes the explicit form

$$
\begin{aligned}
\mathcal{D}_{\mathrm{KL}}\left(\mathcal{N}_1(\mu_1,\Sigma_1)\middle\|\mathcal{N}_2(\mu_2,\Sigma_2)\right) &= \mathbb{E}_{\mathcal{N}_1}[\log\mathcal{N}_1(\mu_1,\Sigma_1) - \log\mathcal{N}_2(\mu_2,\Sigma_2)] \\
&= \mathbb{E}_{\mathcal{N}_1}\Bigg[-\log(2\pi)^{\frac{k}{2}} - \log|\Sigma_1|^{\frac{1}{2}} - \frac{1}{2}(x-\mu_1)^{\mathsf{T}}\Sigma_1^{-1}(x-\mu_1) \\
&\quad - \left(-\log(2\pi)^{\frac{k}{2}} - \log|\Sigma_2|^{\frac{1}{2}} - \frac{1}{2}(x-\mu_2)^{\mathsf{T}}\Sigma_2^{-1}(x-\mu_2)\right)\Bigg] \\
&= \frac{1}{2}\Bigg(\log|\Sigma_2| - \log|\Sigma_1| + \mathbb{E}_{\mathcal{N}_1}\left[-\mathrm{tr}\left((x-\mu_1)^{\mathsf{T}}\Sigma_1^{-1}(x-\mu_1)\right)\right] \\
&\quad + \mathbb{E}_{\mathcal{N}_1}\left[(x-\mu_2)^{\mathsf{T}}\Sigma_2^{-1}(x-\mu_2)\right]\Bigg) \\
&= \frac{1}{2}\Bigg(\log\frac{|\Sigma_2|}{|\Sigma_1|} + \mathbb{E}_{\mathcal{N}_1}\left[-\mathrm{tr}\left(\Sigma_1^{-1}(x-\mu_1)(x-\mu_1)^{\mathsf{T}}\right)\right] \\
&\quad + (\mu_1-\mu_2)^{\mathsf{T}}\Sigma_2^{-1}(\mu_1-\mu_2) + \mathrm{tr}\left(\Sigma_2^{-1}\Sigma_1\right)\Bigg) \\
&= \frac{1}{2}\Bigg(\log\frac{|\Sigma_2|}{|\Sigma_1|} + \mathbb{E}_{\mathcal{N}_1}\left[-\mathrm{tr}\left(\Sigma_1^{-1}\Sigma_1\right)\right] + (\mu_1-\mu_2)^{\mathsf{T}}\Sigma_2^{-1}(\mu_1-\mu_2) \\
&\quad + \mathrm{tr}\left(\Sigma_2^{-1}\Sigma_1\right)\Bigg) \\
&= \frac{1}{2}\Bigg(\log\frac{|\Sigma_2|}{|\Sigma_1|} + \mathbb{E}_{\mathcal{N}_1}\left[-\mathrm{tr}\left(I\right)\right] + (\mu_1-\mu_2)^{\mathsf{T}}\Sigma_2^{-1}(\mu_1-\mu_2) \\
&\quad + \mathrm{tr}\left(\Sigma_2^{-1}\Sigma_1\right)\Bigg) \\
&= \frac{1}{2}\left(\log\frac{|\Sigma_2|}{|\Sigma_1|} - k + (\mu_1-\mu_2)^{\mathsf{T}}\Sigma_2^{-1}(\mu_1-\mu_2) + \mathrm{tr}\left(\Sigma_2^{-1}\Sigma_1\right)\right),
\end{aligned}
$$

where identities from [45] and the cyclic property of the trace are used. In the special case where one of the Gaussians is standard, the KL divergence is

$$
\mathcal{D}_{\mathrm{KL}}\left(\mathcal{N}(\mu,\Sigma)\middle\|\mathcal{N}(0,I)\right) = \frac{1}{2}\left(\mathrm{tr}(\Sigma) + \mu^{\mathsf{T}}\mu - k - \log|\Sigma|\right).
$$

B | Depth-CVAE architecture

B.1 U-Net

	Layer	Layer type	Act	Norm	Input	Output shape
Encoder (E)	0	Input	-	-	-	$3 \times 288 \times 512$
	1	Conv2d 7×7	ReLU	BN	-	$64 \times 144 \times 256$
	2	Max-pool 3×3	-	-	-	$64 \times 72 \times 128$
		BasicBlock $\times 2$	ReLU	BN	-	$64 \times 72 \times 128$
	3	BasicBlock $\times 2$	ReLU	BN	-	$128 \times 36 \times 64$
	4	BasicBlock $\times 2$	ReLU	BN	-	$256 \times 18 \times 32$
	5	BasicBlock $\times 2$	ReLU	BN	-	$512 \times 9 \times 16$
Decoder (D)	0	BasicBlock	ReLU	BN	-	$512 \times 9 \times 16$
	1	Upsample	-	-	-	$512 \times 18 \times 32$
		Conv2d 3×3	-	-	-	$256 \times 18 \times 32$
		Concatenate	-	-	E4	$512 \times 18 \times 32$
		BasicBlock	ReLU	BN	-	$256 \times 18 \times 32$
	2	Upsample	-	-	-	$256 \times 36 \times 64$
		Conv2d 3×3	-	-	-	$128 \times 36 \times 64$
		Concatenate	-	-	E3	$256 \times 36 \times 64$
		BasicBlock	ReLU	BN	-	$128 \times 36 \times 64$
	3	Upsample	-	-	-	$128 \times 72 \times 128$
		Conv2d 3×3	-	-	-	$64 \times 72 \times 128$
		Concatenate	-	-	E2	$128 \times 72 \times 128$
		BasicBlock	ReLU	BN	-	$64 \times 72 \times 128$
	4	Upsample	-	-	-	$64 \times 144 \times 256$
		Conv2d 3×3	-	-	-	$64 \times 144 \times 256$
		Concatenate	-	-	E1	$128 \times 144 \times 256$
		BasicBlock	ReLU	BN	-	$64 \times 144 \times 256$
	5	Upsample	-	-	-	$64 \times 288 \times 512$
		Conv2d 3×3	Sigmoid	-	-	$1 \times 288 \times 512$

Table B.1: *Detailed network architecture for the U-Net stream of the Depth-CVAE. BN is batch normalization, BasicBlock is a residual layer as depicted in Figure B.1 and Upsample layers use bilinear interpolation. The output shape gives information about stride and padding used in convolutional layers. Notice that the encoder is a ResNet-18[6].*

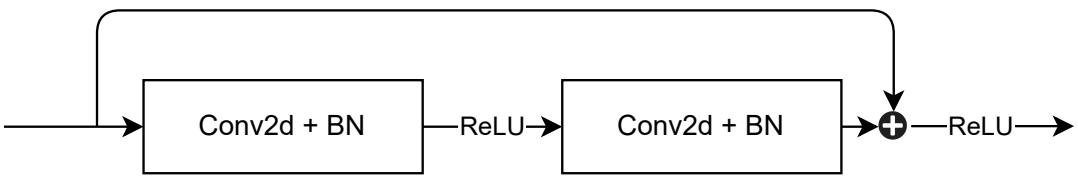


Figure B.1: *BasicBlock: a residual connection with two convolutional layers consisting of 3×3 filters and batch normalization.*

B.2 CVAE

	Layer	Layer type	Act	Norm	Input	Output shape
Encoder	0	Input	-	-	-	$1 \times 288 \times 512$
	1	Conv2d 7×7	ReLU	BN	-	$64 \times 144 \times 256$
		Concatenate	-	-	UNET-D4	$128 \times 144 \times 256$
		BasicBlock	ReLU	BN	-	$64 \times 144 \times 256$
	2	Conv2d 3×3	-	BN	-	$64 \times 72 \times 128$
		Concatenate	-	-	UNET-D3	$128 \times 72 \times 128$
		BasicBlock	ReLU	BN	-	$64 \times 72 \times 128$
	3	Conv2d 3×3	-	BN	-	$128 \times 36 \times 64$
		Concatenate	-	-	UNET-D2	$256 \times 36 \times 64$
		BasicBlock	ReLU	BN	-	$128 \times 36 \times 64$
	4	Conv2d 3×3	-	BN	-	$256 \times 18 \times 32$
		Concatenate	-	-	UNET-D1	$512 \times 18 \times 32$
		BasicBlock	ReLU	BN	-	$256 \times 18 \times 32$
	5	Conv2d 3×3	-	BN	-	$512 \times 9 \times 16$
		Concatenate	-	-	UNET-D0	$1024 \times 9 \times 16$
		BasicBlock	ReLU	BN	-	$512 \times 9 \times 16$
Code	0	Flatten	-	-	-	$1 \times (512 \cdot 9 \cdot 16)$
	1	Linear($\boldsymbol{\mu}$)	-	-	-	1×128
		Linear($\log \boldsymbol{\Sigma}$)	-	-	-	1×128
		Reparameterize($\boldsymbol{z}$)	-	-	-	1×128
		Linear	-	-	-	$1 \times (512 \cdot 9 \cdot 16)$
	2	Reshape	-	-	-	$512 \times 9 \times 16$
Decoder	0	Concatenate	-	-	UNET-D0	$1024 \times 9 \times 16$
		BasicBlock	ReLU	BN	-	$512 \times 9 \times 16$
	1	Upsample	-	-	-	$512 \times 18 \times 32$
		Conv2d 3×3	-	-	-	$256 \times 18 \times 32$
		Concatenate	-	-	UNET-D1	$512 \times 18 \times 32$
		BasicBlock	ReLU	BN	-	$256 \times 18 \times 32$
	2	Upsample	-	-	-	$256 \times 36 \times 64$
		Conv2d 3×3	-	-	-	$128 \times 36 \times 64$
		Concatenate	-	-	UNET-D2	$256 \times 36 \times 64$
		BasicBlock	ReLU	BN	-	$128 \times 36 \times 64$
	3	Upsample	-	-	-	$128 \times 72 \times 128$
		Conv2d 3×3	-	-	-	$64 \times 72 \times 128$
		Concatenate	-	-	UNET-D3	$128 \times 72 \times 128$
		BasicBlock	ReLU	BN	-	$64 \times 72 \times 128$
	4	Upsample	-	-	-	$64 \times 144 \times 256$
		Conv2d 3×3	-	-	-	$64 \times 144 \times 256$
		Concatenate	-	-	UNET-D4	$128 \times 144 \times 256$
		BasicBlock	ReLU	BN	-	$64 \times 144 \times 256$
	5	Upsample	-	-	-	$64 \times 288 \times 512$
		Conv2d 3×3	Sigmoid	-	-	$1 \times 288 \times 512$

Table B.2: *Detailed network architecture for the CVAE stream of the Depth-CVAE. BN is batch normalization, BasicBlock is a residual layer as depicted in Figure B.1 and Upsample layers use bilinear interpolation. The output shape gives information about stride and padding used in convolutional layers. The linear layers in the code are separate,* ***not*** *subsequent, and each predict* $\boldsymbol{\mu}$ *or* $\log \boldsymbol{\Sigma}$*.*

C | Motion autoencoder architecture

	Layer	Layer type	Act	Norm	Output shape
Encoder	0	Input	-	-	$3 \times 375 \times 1242$
	1	Conv2d 7×7	ReLU	BN	$64 \times 187 \times 621$
	2	Max-pool 3×3	-	-	$64 \times 93 \times 310$
		BasicBlock $\times 2$	ReLU	BN	$64 \times 93 \times 310$
	3	BasicBlock $\times 2$	ReLU	BN	$128 \times 46 \times 155$
	4	BasicBlock $\times 2$	ReLU	BN	$256 \times 23 \times 77$
	5	BasicBlock $\times 2$	ReLU	BN	$512 \times 11 \times 38$
Decoder	0	BasicBlock	ReLU	BN	$256 \times 11 \times 38$
		Upsample	-	BN	$256 \times 22 \times 76$
	1	BasicBlock	ReLU	BN	$128 \times 22 \times 76$
		Upsample	-	BN	$128 \times 44 \times 152$
	2	BasicBlock	ReLU	BN	$64 \times 44 \times 152$
		Upsample	-	BN	$64 \times 88 \times 304$
	3	BasicBlock	ReLU	BN	$64 \times 88 \times 304$
		Upsample	-	BN	$64 \times 176 \times 608$
	4	BasicBlock	ReLU	BN	$1 \times 176 \times 608$
		Upsample	Sigmoid	BN	$1 \times 352 \times 1216$

Table C.1: *Detailed network architecture for the motion autoencoder. BN is batch normalization, BasicBlock is a residual layer as depicted in Figure B.1 and Upsample layers use transposed convolutions with 2×2 filters. The output shape gives information about stride and padding used in convolutional layers. Notice that the encoder is a ResNet-18[6].*

D | RefineNet architecture

	Layer	Layer type	Act	Norm	Input	Output shape
Encoder (E)	0	Input	-	-	-	$3 \times 288 \times 512$
	1	Conv2d 7×7	ReLU	BN	-	$64 \times 144 \times 256$
		BasicBlock	ReLU	BN	-	$64 \times 144 \times 256$
	2	BasicBlock	ReLU	BN	-	$128 \times 72 \times 128$
	3	BasicBlock	ReLU	BN	-	$256 \times 36 \times 64$
	4	BasicBlock	ReLU	BN	-	$512 \times 18 \times 32$
Decoder (D)	0	BasicBlock	ReLU	BN	-	$512 \times 18 \times 32$
	1	Upsample	-	-	-	$512 \times 36 \times 64$
		Conv2d 3×3	-	-	-	$256 \times 36 \times 64$
		Concatenate	-	-	E3	$512 \times 36 \times 64$
		BasicBlock	ReLU	BN	-	$256 \times 36 \times 64$
	2	Upsample	-	-	-	$256 \times 72 \times 128$
		Conv2d 3×3	-	-	-	$128 \times 72 \times 128$
		Concatenate	-	-	E2	$256 \times 372 \times 128$
		BasicBlock	ReLU	BN	-	$128 \times 72 \times 128$
	3	Upsample	-	-	-	$128 \times 144 \times 256$
		Conv2d 3×3	-	-	-	$64 \times 144 \times 256$
		Concatenate	-	-	E1	$128 \times 144 \times 256$
		BasicBlock	ReLU	BN	-	$64 \times 144 \times 256$
	4	Upsample	-	-	-	$64 \times 288 \times 512$
		Conv2d 3×3	Sigmoid	-	-	$1 \times 288 \times 512$

Table D.1: *Detailed network architecture for RefineNet. BN is batch normalization, BasicBlock is a residual layer as depicted in Figure B.1 and Upsample layers use bilinear interpolation. The output shape gives information about stride and padding used in convolutional layers.*

E | Additional results

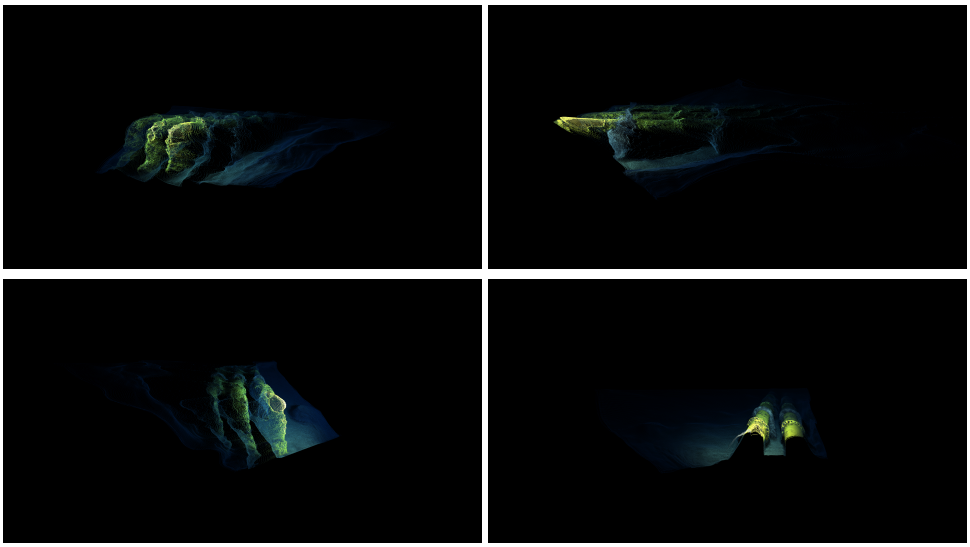


Figure E.1: *Additional results from the baseline Depth-CVAE with RGB input.*

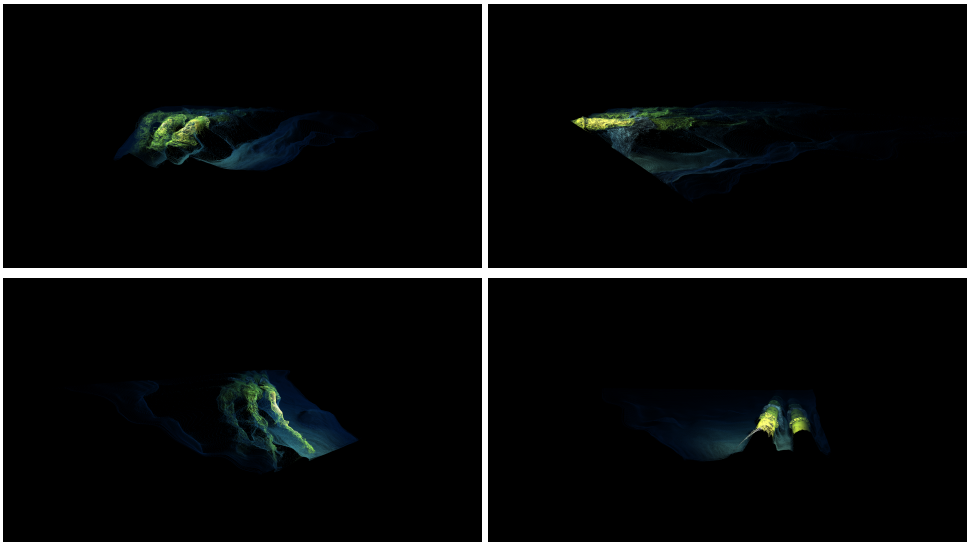


Figure E.2: *Additional results from the baseline Depth-CVAE with grayscale input.*

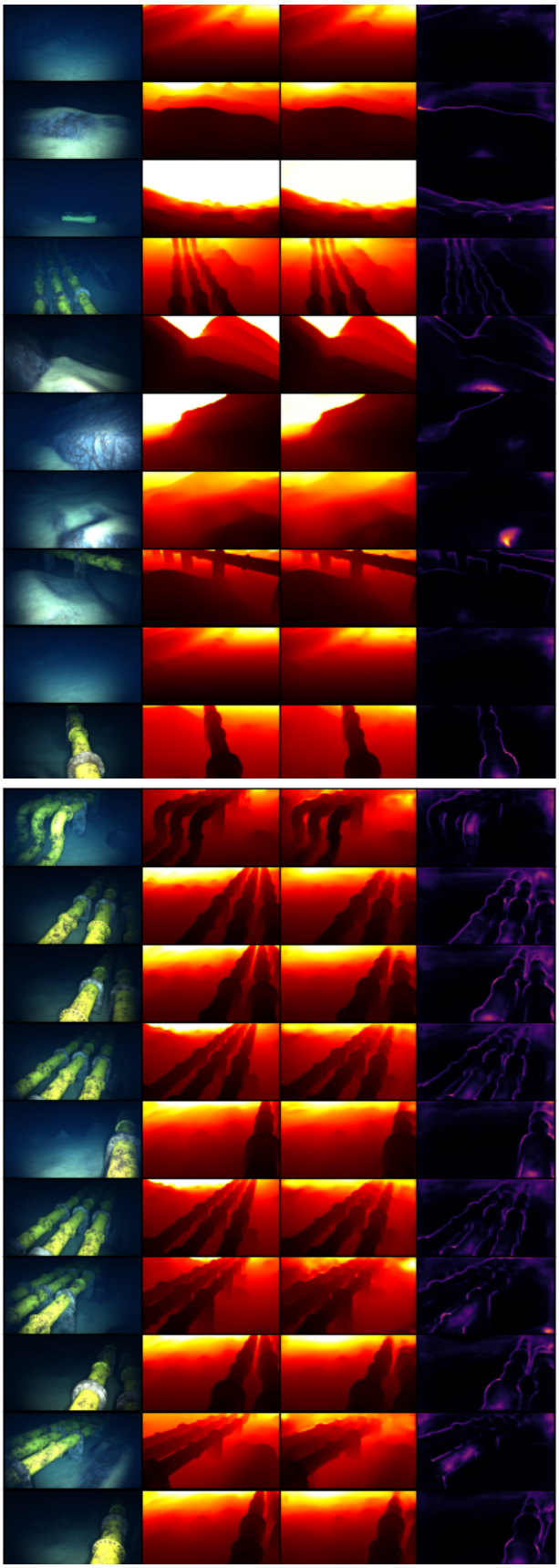


Figure E.3: *Additional results from the VAROS train set (top) and test set (bottom). From left to right: RGB image, ground truth depth, predicted depth and uncertainty of predicted depth. Brighter areas indicate higher values.*

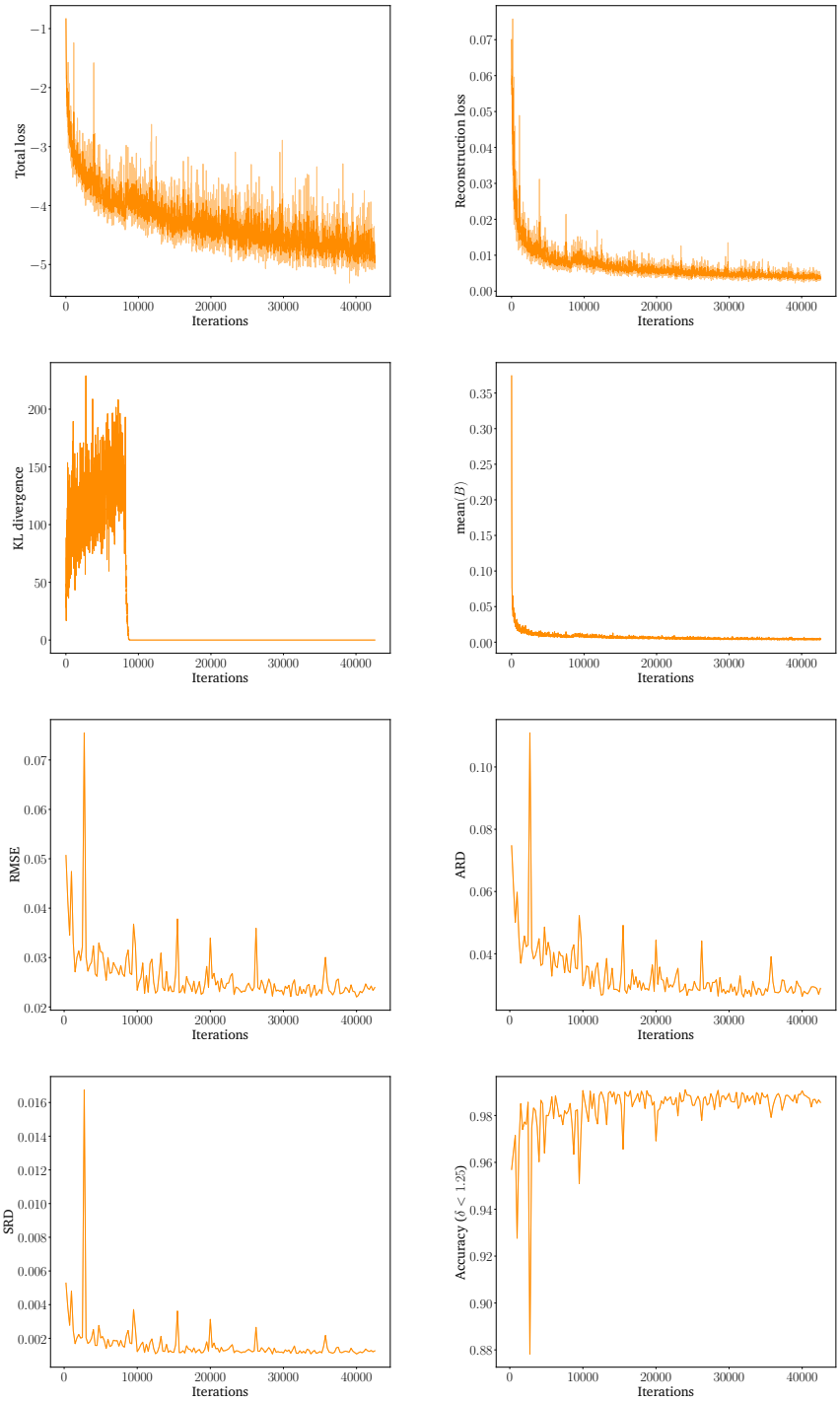


Figure E.4: *Losses and metrics during training for the baseline Depth-CVAE with RGB input. B is the uncertainty map predicted by the model.*

NTNU
Norwegian University of
Science and Technology

aros vision group