NTNU

Kunnskap for en bedre verden

DEPARTMENT OF ENGINEERING CYBERNETICS

Deep Monocular Depth Estimation for Autonomous Underwater Vehicles

Anders Thallaug Fagerli

Supervisor: Annette Stahl
Co-supervisor: Mauhing Yip

20th December 2021

Abstract

This report considers the estimation of depth from monocular images using methods based on deep learning, and is aimed at producing the structure of the underwater environment for an autonomous underwater vehicle. Depth estimation, despite being a popular research topic, has not been extensively studied for underwater operations, and therefore motivates further investigation in this area.

A review of the current state-of-the-art methods for estimating depth from monocular scenes is given, with the aim of forming an overview of different architectures and design choices popularly used for depth estimation. The variational autoencoder is chosen among the methods as a promising model for the underwater environment, and is implemented for testing on two synthetic datasets.

Results from the datasets show promising results, displaying the predictive performance of the variational autoencoder, and simultaneously the strength of using the estimated uncertainty of the predictions for further reasoning. There are, however, shortcomings in the trained model, and results from real underwater images indicate that there is still more to learn.

Sammendrag

Denne rapporten tar for seg estimering av dybde fra monokulære bilder ved bruk av metoder basert på dyp læring, og er rettet mot å produsere strukturen til undervannsmiljøet for et autonomt undervannsfartøy. Dybdeestimering, til tross for at det er et populært forskningstema, har ikke blitt grundig studert for undervannsoperasjoner, og motiverer derfor videre undersøkelser på dette området.

Det gis en gjennomgang av dagens toppmoderne metoder for å estimere dybde fra monokulære bilder, med hensikten å danne en oversikt over ulike arkitekturer og designvalg populært brukt for dybdeestimering. Variasjonsautokoderen er valgt blant metodene som en lovende modell for undervannsmiljøet, og er implementert for testing på to syntetiske datasett.

Resultater fra datasettene viser lovende resultater, som viser den prediktive ytelsen til den variasjonelle autokoderen, og samtidig styrken ved å bruke den estimerte usikkerheten til prediksjonene for videre resonnering. Det er imidlertid mangler ved den trente modellen, og resultater fra ekte undervannsbilder indikerer at det fortsatt er mye å lære.

Preface

The following work is the result of the specialization project, TTK4550, for the study program Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU), counting for 15 credit points. The work is conducted for the Autonomous Robots for Ocean Sustainability (AROS) project, supported by the Research Council of Norway (project number: 304667).

I would like to thank my supervisors, Annette Stahl and Mauhing Yip, and the rest AROS Vision Group, for giving me this exciting project and for valuable guidance throughout the semester. It is truly inspiring to work with such dedicated people, and to finally work on a real-world problem has been an extra motivation for my final year at NTNU.

Anders Thallaug Fagerli,
Trondheim, December 2021

Table of Contents

List of Figures

List of Abbreviations

AROS Autonomous Robots for Ocean Sustainability
AUV Autonomous underwater vehicle

CNN Convolutional neural network
CVAE Conditional variational autoencoder

GAN Generative adversarial network

KL Kullback-Leibler

MSE Mean squared error

SFM Structure from motion
SGVB Stochastic gradient variational bayes
SLAM Simultaneous localization and mapping

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 high risk of human lives. The Autonomous Robots for Ocean Sustainability (AROS) project aims to replace human operators in underwater operations, like visual inspection of man-made structures in underwater environments, as seen in Figure 1.1, where the visual perception of the robot plays a vital role. Mapping the environment is an essential part of any robots perceptual system, for many reasons, but in this work the target is a full (dense) map which can be used for e.g visual inspection and collision avoidance, reconstructing the geometry of the scene by visual information from cameras.

1.1 Problem description

The objective of this report is rooted in the objective of AROS, and more specifically investigating methods for densly mapping the environment of an autonomous underwater vehicle (AUV) using monocular cameras. Estimating structure from individual images is inherently ill-posed[1], but deep learning methods have recently shown great promise even in these cases. Although they perform well on the popular above-water datasets, it remains to see how they tackle more challenging data. This motivates an investigation into how deep learning methods perform in underwater environments, and if they can be modified to achieve similar performance as in the above-water scenarios. The following subtasks are therefore considered for this project:

1. A literature study on related work, state-of-the-art methods and theory for dense depth estimation based on deep learning for monocular images.
2. Implementation of a deep learning-based model for predicting dense depth from images.
3. Verification and testing of the implemented method on datasets realistically relating to the targeted underwater environment.

To limit the scope of the project, considerations such as real-time performance and handling visual degradation in images from underwater environments are not investigated. These are both important considerations, especially the latter, which heavily affects the performance of the methods discussed in this report, and they are therefore left as possible future work.

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

1.2 Contributions

The contributions of this project are in summary:

- The first open-source implementation (as far as the author knows) of a variational autoencoder for depth estimation, located at: `https://github.com/andersfagerli/Depth-CVAE`.
- A review and comparison of different state-of-the-art methods for depth estimation using deep learning, with respect to conditions and constraints imposed on the AROS project.
- An evaluation of the VAROS Synthetic Underwater dataset for deep learning-based methods.

1.3 Thesis outline

The structure of the report reflects the proposed subtasks, and is therefore outlined as follows.

Chapter 2 briefly presents the related work, which is further expanded on in Chapter 3 and Chapter 4. Chapter 3 explains theory and methods for single-view methods, while Chapter 4 describes the multi-view counterpart. In Chapter 5, a choice is made for the specific model to be tested, with implementation details of the chosen model. The results from testing the model on two datasets are presented in Chapter 6, with accompanying discussion. Finally, the report is concluded in Chapter 7, with suggestions for further work.

2 | Related work

Depth estimation from images has been a heavily researched topic for numerous years, early on mainly by inferring the structure of the scene 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 greatest focus, as estimating depth from individual images is ill-posed and not possible without geometric priors of the scenes[1], and is also the focus of this work. The reviewed and related work is therefore limited to monocular methods that utilize deep learning to estimate depth.

Eigen *et al.*[2] 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[3].

Laina *et al.*[4] used the residual connections of [5] 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. Networks using a combination of contracting and expanding paths are typically called *autoencoders*, and are frequently seen in dense prediction tasks such as depth estimation or segmentation.

In Wofk *et al.*[6], a combination of the feature concatenation in [2] and autoencoder structure of [4] 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 by pruning away redundant parameters.

Different from the previously described methods, Ranftl *et al.*[7] 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 [6]. The model outperformed all others on the popular datasets, and is the current state-of-the-art for depth estimation.

Generative models have also shown competing performance in inference tasks as depth estimation, even though they are primarily targeted at data generation. Bloesch *et al.*[8] used a variational autoencoder conditioned on the deep features of the images, generating depth maps from a compact latent representation. A key contribution is the latent representation itself, being a compressed and lower-dimensional representation of the depth maps, which exhibits useful

properties for further dense 3D reconstruction or visual simultaneous localization and mapping (VSLAM).

The works described so far have predicted depth from individual images, and have not used information from multiple views. In Godard *et al.*[9], 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.*[10], two subsequent autoencoders were used to predict the disparity maps instead of depth directly, and by training on stereo images, a virtual stereo term was adopted to improve the accuracy in monocular odometry.

3 | Single-view depth estimation

In this chapter, an overview is given of the different architectures and theory for deep depth estimation using individual images, mainly using CNN-based architectures in a supervised fashion. Theory for autoencoder-like architectures are first presented, before Transformers and their adaptation to depth estimation is shown. Finally, generative models for depth estimation are discussed, with a focus on variational autoencoders.

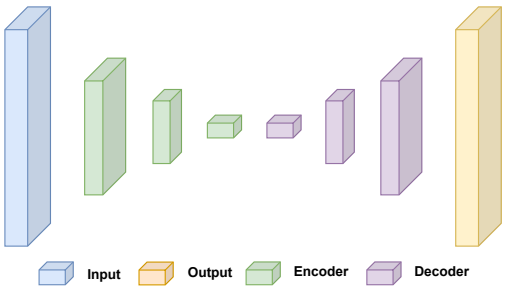
3.1 Autoencoders

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[5], [11], [12] and is used to capture global context, while the decoder is used for enabling fine-grained predictions, often in the same resolution as the input. A problem is that local context, or localization of high resolution features, is lost in the contracting path, giving coarse predictions, and 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 [3] for segmentation, but already in [2] an autoencoder-like CNN for depth estimation was used in the same manner. The most prevalent implementations using this structure for depth estimation are [2] and [4].

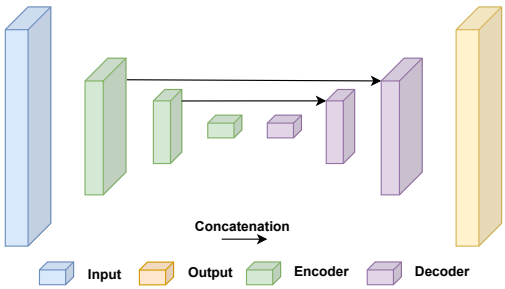
3.1.1 Coarse-to-fine prediction

A vanilla autoencoder, as shown in Figure 3.1a, is typically used to learn encodings of the data, and not for inference directly. The encoding, typically called the *latent representation*, is a lower dimensional feature which can be interpreted as a compressed version of the input, which is useful for reducing the number of features that describe the data, but this in the domain of dimensionality reduction, and not directly useful for inferring e.g 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 3.3.1 where it is of more relevance.

Autoencoders are more commonly used for fine-grained predictions, either for classification or regression. 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 3.1b. This is usually done by concatenating feature maps channel-wise in the decoder, as displayed in Figure 3.2a. This increases the number of channels in the respective decoder layer,

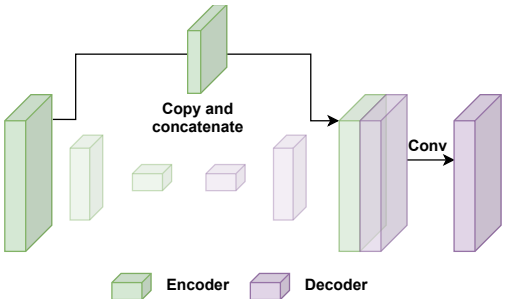


(a) Vanilla autoencoder composed of encoder and decoder.

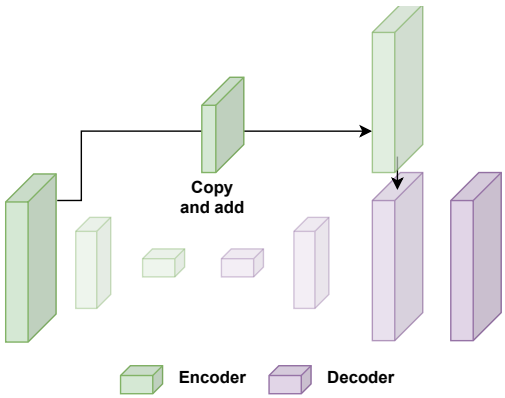


(b) Autoencoder with connections between encoder and decoder for fine-grained predictions.

Figure 3.1: *Graphical representations of CNN-based autoencoders.*



(a) Concatenation of feature maps.



(b) Addition of feature maps.

Figure 3.2: *Different connection types between encoder and decoder.*

and a subsequent convolution is usually performed to reduce the channels afterwards, producing the final decoder feature map. An alternative is to instead add feature maps, as in Figure 3.2b. 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.

3.1.2 Loss functions

There are a variety of different loss functions used in the literature for evaluating the estimated depth, ranging from a simple ℓ_2 loss to more involved combinations using geometric constraints as well. As this chapter is focused on supervised methods, the following loss functions will assume that ground truth depth is available, and are applicable to all supervised depth estimation architectures, and will therefore not be repeated in the subsequent chapters. In the following, y is denoted as the ground truth depth and $\tilde{y}$ is denoted as the estimated depth, each at a single pixel in the depth map.

ℓ_1 loss

The ℓ_1 loss,

$$\mathcal{L}_1(y, \tilde{y}) = |y - \tilde{y}|, \tag{3.1}$$

measures the absolute distance between ground truth and prediction.

ℓ_2 loss

The ℓ_2 loss is perhaps the most common loss function in regression, and is defined as

$$\mathcal{L}_2(y, \tilde{y}) = \|y - \tilde{y}\|_2^2, \tag{3.2}$$

aimed at minimizing the squared euclidean norm between the ground truth and prediction. Although it is common for regression, it is not commonly used in depth estimation.

Huber loss

A robust regression loss, the Huber loss is a weighted combination of ℓ_1 and ℓ_2 that is less sensitive to outliers,

$$\mathcal{L}_\delta(y, \tilde{y}) = \begin{cases} \frac{1}{2}\|y - \tilde{y}\|_2^2 & |y - \tilde{y}| \leq \delta, \\ \delta(|y - \tilde{y}| - \frac{1}{2}\delta) & \text{otherwise}, \end{cases} \tag{3.3}$$

where δ is a parameter to be chosen.

Reverse Huber loss

The reverse Huber loss,

$$\mathcal{B}_\delta(y, \tilde{y}) = \begin{cases} \frac{1}{2}|y - \tilde{y}| & |y - \tilde{y}| \leq \delta, \\ \delta(\|y - \tilde{y}\|_2^2 - \frac{1}{2}\delta) & \text{otherwise}, \end{cases} \tag{3.4}$$

is just the reverse ordering of the Huber loss, also parameterized by δ.

Edge-aware smoothness loss

The edge-aware smoothness loss,

$$\mathcal{L}_{\text{smooth}}(D, I) = \sum_{\boldsymbol{u} \in \Omega} |\nabla_x D| e^{-|\nabla_x I|} + |\nabla_y D| e^{-|\nabla_y I|}, \tag{3.5}$$

is a regularizer used in combination with one of the above loss functions, ensuring that the predicted depth map D is locally smooth by minimizing the gradients along the x and y-axis, weighted by the image gradients, at every pixel $\boldsymbol{u}$.

3.1.3 Autoencoders for depth estimation

The main ideas from the previous chapters are using an expanding decoder and concatenating features, which both can be used for estimating depth from images, either by themselves or together. The seminal work of [2] used a contracting path to predict a coarse depth map, and refined it by concatenating it to a finer contracting path with higher resolution, giving both local and global context in the final prediction. In [4], an autoencoder without connections between the encoder and decoder was used, and instead increased the number of layers substantially by using the residual blocks in [5], using the deeper connections to predict local information from global context. A combination is used in [6], where the connections between the encoder and decoder are performed by addition, rather than concatenation, giving better computational performance while still retaining information. Common for all these works is the autoencoder

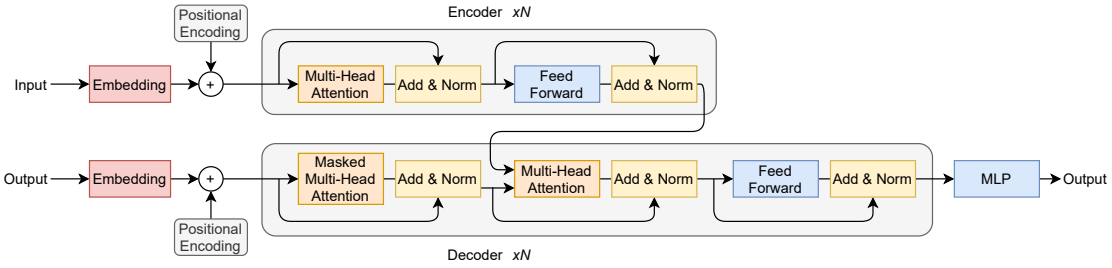


Figure 3.3: *Blocks of a Transformer, using N encoder and decoder blocks, and a multilayer perceptron at the output. The figure is highly inspired by [13].*

(or autoencoder-like) structure for estimating depth, a structure that will be seen in the coming chapters as well.

3.2 Transformers

The Transformer, first introduced in [13], is a new neural network architecture that uses an *attention* mechanism instead of the popularly used convolutions, and is therefor entirely non-convolutional. Its main use has been for natural language processing, in tasks such as translation, but has recently been adapted to computer vision as well[14]. At its core is attention, which has a global receptive field, in contrast to convolutions, which need multiple stages to infer global context. A generic example of its architecture is shown in Figure 3.3. As this is an entirely new architecture, each block will be described in the following, with emphasis on the attention block.

3.2.1 Architecture

As many dense prediction networks, the Transformer uses an encoder-decoder architecture for densely mapping the input to a high resolution output. Following this structure, the Transformer first encodes input data $\boldsymbol{x} = (x_1, ..., x_n)$ to a latent representation $\boldsymbol{z} = (z_1, ..., z_n)$, and then decodes it into the output $\boldsymbol{y} = (y_1, ..., y_m)$. Notice that the dimensionality of the latent representation is the same as the input, but the output may be of another dimensionality. In Transformer terminology, each data point is referred to as a *token*. Using the example of natural language processing, a token is a word in a sentence, so e.g the sentence

> 'Can Transformers be used for depth estimation?'

has the tokens 'Can', 'used', '?', etc.

Embedding and positional encoding

The input data is first embedded into a vector representation using numbers, before it is passed to the encoder. In the above sentence, the token 'Can' may e.g have $\boldsymbol{t}_1 = [0.1, 0.34, 0.45]^\mathsf{T}$ as embedding. Typically, the embeddings themselves are learned by the model through some linear transformation, which results in tokens of dimension d_{model}.

For the Transformer to use information about the order of the sequence of data, we need to give it relative or absolute position of each token. This is implemented by concatenating an additional positional token to the original token.

Encoder and decoder stacks

The encoder consists of N identical blocks, where each block has two components. The first is the self-attention mechanism and the second is a fully connected feed-forward layer. Additionally, there is a residual connection[5] around each of these components, and normalization after each.

The decoder also has N identical blocks and follows the exact structure of the encoder, but has an additional attention layer that takes as input the output of the encoder stack. The architecture shown in Figure 3.3 is specialized towards natural language processing, and has elements for this specific task, such as the masked multi-head attention and output as additional input to the decoder. This is not the case for the vision-adapted Transformers, and these elements will therefore not be expanded on here.

Attention

At the core of the Transformer, this operation computes the relationship between the different tokens and scores them based on how they are related. For each token $\boldsymbol{t}_i$, we create vectors query $\boldsymbol{q}_i$, key $\boldsymbol{k}_i$ and value $\boldsymbol{v}_i$. Each of these vectors are computed from a linear transformation of the token, such that

$$\boldsymbol{q}_i = \mathbf{W}^q \boldsymbol{t}_i \in \mathbb{R}^{d_k}, \tag{3.6a}$$

$$\boldsymbol{k}_i = \mathbf{W}^k \boldsymbol{t}_i \in \mathbb{R}^{d_k}, \tag{3.6b}$$

$$\boldsymbol{v}_i = \mathbf{W}^v \boldsymbol{t}_i \in \mathbb{R}^{d_v}, \tag{3.6c}$$

where each weighting matrix $\mathbf{W}$ is a learned transformation. Note that the weights are shared for all tokens, and the queries and keys are of same dimension. For a neural network, these matrices correspond to the weights of a linear layer.

The tokens are then scored against each other by taking the inner product of the query of one token with the key of another, $\text{score}_{ij} = \boldsymbol{q}_i^\mathsf{T} \boldsymbol{k}_j$. Each token will then get a score for each of the other tokens, where the score tells how much they relate to each other. All the scores for each token, $\text{scores}_i = (\text{score}_{i1}, \text{score}_{i2}, ..., \text{score}_{in})$, are then normalized by a Softmax such that they all sum to one. The value $\boldsymbol{v}_i$ for each token is then multiplied by the normalized scores, producing n value vectors for each token. The n value vectors are then summed to produce the final value vector for a single token, and this is done for every token. The final value vector gives the value of the token compared to all the other, and gives information about how important the single token is in the global context.

In practice, the attention for multiple tokens is computed simultaneously by packing the vectors into matrices, so e.g the query vectors are packed into $\mathbf{Q} = [\boldsymbol{q}_1^\mathsf{T}, \boldsymbol{q}_2^\mathsf{T}, ..., \boldsymbol{q}_n^\mathsf{T}]^\mathsf{T}$. This is also done for the keys and values, giving matrices $\mathbf{K}$ and $\mathbf{V}$. The output of the attention operation can then be written

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\mathsf{T}}{\sqrt{d_k}}\right)\mathbf{V}, \tag{3.7}$$

producing an output of dimension $n \times d_v$, and $1 \times d_v$ after summing the n vectors. Before normalizing the scores, it is common to scale each score by $\sqrt{d_k}$ as in (3.7), as this gives better gradient flow during backpropagation[13].

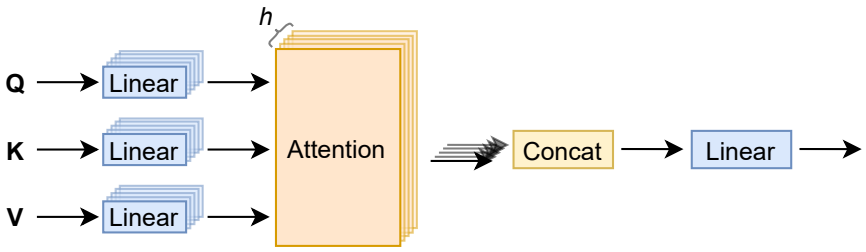


Figure 3.4: *Multi-head attention using h attention heads.*

A problem is that the weights are shared for all tokens, inhibiting the model to learn more complex relationships. A solution is to use several attention blocks in parallel, called multi-head attention, as shown in Figure 3.4. The multi-head attention operation is then

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, ..., \text{head}_h)\mathbf{W}^O, \tag{3.8a}$$

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V), \tag{3.8b}$$

where h is the number of heads and the projection matrices have dimensions $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$, $\mathbf{W}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $\mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$.

Feed forward

The output of the attention block is forwarded through a small, fully connected layer that has an optional number of hidden layers and non-linearities, producing the final encoder block output for each token of dimension d_{model}. At the last encoder block, we have the latent representation for each token, $\boldsymbol{z}$, which is forwarded to the decoder.

3.2.2 Transformers for depth estimation

The Transformer was first used for a vision-related task in [14], namely for classification, and further adapted to depth estimation in [7], which is the current state-of-the-art. As it is not clear from the previous chapter how a Transformer may be used to infer depth, a brief description is given in the following.

The main idea of [14] is to divide the image into smaller, non-overlapping patches that are embedded into tokens. This is a bag-of-words[15]-like mechanism, where each patch (or token) is a visual word in the bag. This embedding is, as previously, done by a learned linear projection. The tokens are then forwarded through multiple encoder layers, each computing the attention for each token. The Transformer in [7] does, however, not use a decoder similar to Figure 3.3, but rather something closer to the decoder described in Chapter 3.1. The overall architecture is shown in Figure 3.5. After each encoder block, the tokens are concatenated to form an image-like feature map, and then upsampled to produce an image at a higher resolution, which is at last fused with the feature map from the decoder stream by addition. The decoder stream does not have any attention mechanism, and only uses convolutions and upsampling, as in Chapter 3.1. At last is a task-specific output head, which computes the final depth values.

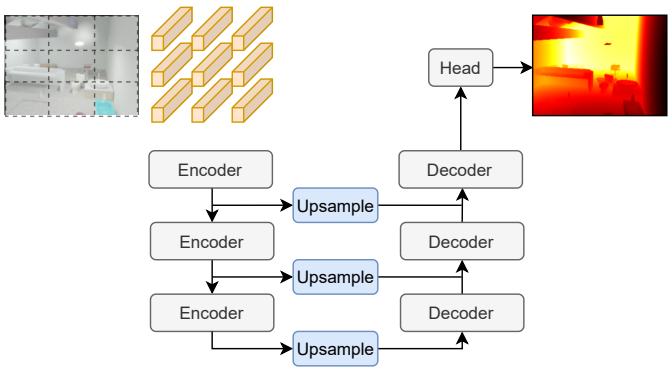


Figure 3.5: *Architecture of a Transformer predicting depth from an image. The image is divided into patches and embedded into tokens (orange), before the tokens are forwarded through multiple encoder blocks. Latent representations after each encoder block are concatenated and upsampled to produce feature maps, which are further fused with the convolutional decoder.*

3.3 Generative models

Generative models have in recent works been shown to produce depth maps of competing performance to the preceding methods [8], [16]–[19]. Although these models are primarily targeted at *generation* of data, they also possess useful properties for inference. Of these models, it is mainly generative adversarial networks (GANs) and variational autoencoders (VAEs) that have shown success in depth estimation. To limit the scope, only VAEs are considered in this chapter, and the reader is referred to [16] for more details on how GANs can be used for depth estimation.

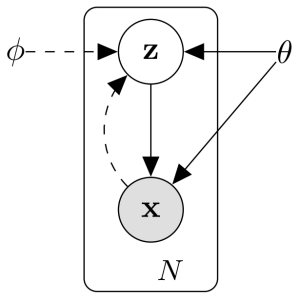
3.3.1 Variational autoencoders

First introduced by [20], and later adapted for depth estimation by [8], the 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, we see how the 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.

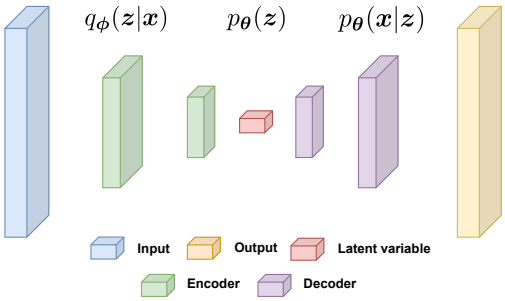
Variational inference with intractable latent posteriors

The motivation of [20] is to solve inference problems where we have 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, we want to sample new data $\boldsymbol{x}$ from the distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, so in this case our objective is to find $p_{\boldsymbol{\theta}}(\boldsymbol{x})$. Using the law of total probability,

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z}) d\boldsymbol{z}, \tag{3.9}$$



(a) *Bayesian network of the model [20]. 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 $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$.*



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

Figure 3.6: *Different graphical representations of the directed probabilistic model under consideration.*

we see that 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 (3.9). The idea is to sample $\boldsymbol{z}$ that produce $\boldsymbol{x}$, meaning we need some distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ to give the $\boldsymbol{z}$ that are likely under $\boldsymbol{x}$. Using Bayes' rule,

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}, \tag{3.10}$$

we see that this distribution is in the general case intractable to solve, as it includes the intractable (3.9). 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 3.6a.

Relating the above to neural networks, we denote $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ 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 we sample $\boldsymbol{z}$ from, where $\boldsymbol{z}$ is typically denoted as the *code*. Figure 3.6b 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

As we desire a model we can sample new data from, we want to find the parameters that maximizes the marginal likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x})$. Assuming we have a 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

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(N)}) = \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}). \tag{3.11}$$

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

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) = \mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) \middle|\middle| p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})\right) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)}), \tag{3.12}$$

where

$$\mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})\middle\|p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})\right)=\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})-\log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})\right] \tag{3.13}$$

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, we have $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \geq \mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi},\boldsymbol{x}^{(i)})$, 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

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi},\boldsymbol{x}^{(i)}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[-\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})+\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})+\log p_{\boldsymbol{\theta}}(\boldsymbol{z})\right]\\ &= -\mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})\middle\|p_{\boldsymbol{\theta}}(\boldsymbol{z})\right)+\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\right].\end{aligned} \tag{3.14}$$

Our objective is to maximize (3.11), but since the KL divergence in (3.11) involves the unknown $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$, we cannot maximize it directly. Instead, we estimate the marginal log-likelihood by the variational lower bound, as this involves distributions we can control, and instead maximize this. We then need to specifically define the distributions $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$.

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

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \equiv \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}),\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x})), \tag{3.15a}$$

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}) \equiv \mathcal{N}(\mathbf{0},\mathbf{I}), \tag{3.15b}$$

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) \equiv \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}),\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z})), \tag{3.15c}$$

where $\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x})$ is diagonal. For a neural network, as in Figure 3.6b, $\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x})$ 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 (3.15) are simplifying, they allow for efficient computations, and turn out to give good performance. Having defined the distributions, the objective in (3.14) 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[20], but using the distributions defined in (3.15) it takes the explicit form

$$\mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})\middle\|p_{\boldsymbol{\theta}}(\boldsymbol{z})\right)=\frac{1}{2}\left(\mathrm{tr}(\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)}))+\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})^{\mathsf{T}}\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})-k-\log|\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{x}^{(i)})|\right), \tag{3.16}$$

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,

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\right] \approx \frac{1}{L}\sum_{l=1}^{L}\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i,l)}), \tag{3.17}$$

but this is computationally expensive. Instead, we approximate the expectation by $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i)})$, where $\boldsymbol{z}^{(i)}$ is a single sample corresponding to $\boldsymbol{x}^{(i)}$. This is usually a poor approximation, but since we are averaging over a batch of data when training the model, the approximation holds.

Using the distributions defined in (3.15), we get

$$\begin{aligned}\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\right] &\approx \log \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)}), \boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})\right) \\ &= -\frac{k}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})| - \frac{1}{2}\left\|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})\right\|^2_{\boldsymbol{\Sigma}_{\boldsymbol{x}}}, \end{aligned} \tag{3.18}$$

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

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

$$\begin{aligned}\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)}) &= -\mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})\middle\|p_{\boldsymbol{\theta}}(\boldsymbol{z})\right) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i)}) \\ &= -\frac{1}{2}\left(\mathrm{tr}(\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})^{\mathsf{T}}\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)}) - k - \log|\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})|\right) \\ &\quad + \left(-\frac{k}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})| - \frac{1}{2}\left\|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})\right\|^2_{\boldsymbol{\Sigma}_{\boldsymbol{x}}}\right), \end{aligned} \tag{3.19}$$

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

$$\tilde{\mathcal{L}}^M(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathbf{X}^M) = \frac{1}{M}\sum_{i=1}^{M}\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)}). \tag{3.20}$$

The optimal parameters, and thus the distributions, can then be found by maximizing (3.20) for a sufficient amount of minibatches M, as an approximation to (3.11).

A problem when calculating the gradients of (3.20) 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 a solution is to reparameterize $\boldsymbol{z}^{(i)}$ by the differentiable transformation

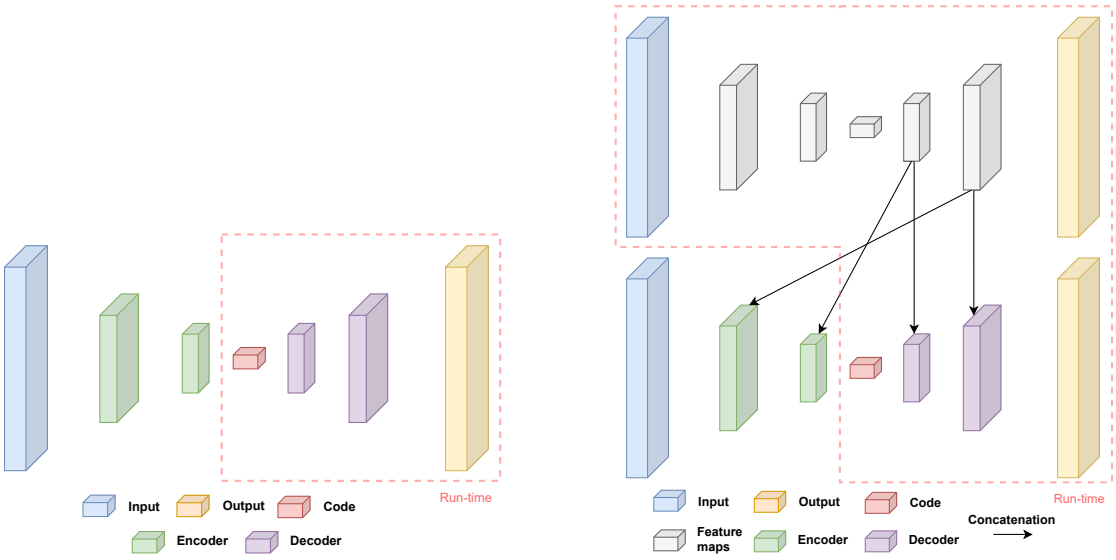
$$\boldsymbol{z}^{(i)} = \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)}) + \boldsymbol{\Sigma}_{\boldsymbol{z}}^{\frac{1}{2}}(\boldsymbol{x}^{(i)}) \cdot \boldsymbol{\epsilon}, \tag{3.21}$$

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 (3.21) is just another way of writing $\boldsymbol{z}^{(i)}$ given from (3.15a), 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 (3.11) can finally be solved by finding

$$\begin{aligned}\boldsymbol{\theta}^*, \boldsymbol{\phi}^* &= \underset{\boldsymbol{\theta},\boldsymbol{\phi}}{\mathrm{argmax}}\,\frac{1}{M}\sum_{i=1}^{M}\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)}) \\ &= \underset{\boldsymbol{\theta},\boldsymbol{\phi}}{\mathrm{argmax}}\,\frac{1}{M}\sum_{i=1}^{M} -\frac{1}{2}\left(\mathrm{tr}(\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})^{\mathsf{T}}\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)}) - k - \log|\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})|\right) \\ &\qquad + \left(-\frac{1}{2}\log|\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})| - \frac{1}{2}\left\|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})\right\|^2_{\boldsymbol{\Sigma}_{\boldsymbol{z}}}\right) \\ &= \underset{\boldsymbol{\theta},\boldsymbol{\phi}}{\mathrm{argmin}}\,\frac{1}{M}\sum_{i=1}^{M}\frac{1}{2}\left(\mathrm{tr}(\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})^{\mathsf{T}}\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)}) - k - \log|\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}^{(i)})|\right) \\ &\qquad + \left(\frac{1}{2}\log|\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})| + \frac{1}{2}\left\|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}^{(i)})\right\|^2_{\boldsymbol{\Sigma}_{\boldsymbol{z}}}\right), \end{aligned} \tag{3.22}$$

where the constant terms in (3.19) 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, we can repeatedly draw



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

Figure 3.7: *Possible CNN implementations of a VAE (a) and a CVAE (b).*

minibatches from the whole dataset to find the optimal parameters of the whole dataset.

Note also the two terms in (3.22) we aim to minimize, specifically the KL divergence and the *reconstruction error*. The reconstruction error is the direct measure of how well the model reconstructs the data, which we naturally want to 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 isn't 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 (3.15). 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 couldn't 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 3.7a, 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.

Conditional variational autoencoders

A limitation of the VAE is that we have no control over the data it generates. If it is trained on generating new numbers from the MNIST dataset [21], a random code will give a random digit,

which means we can't generate specific digits. 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) [22] solves this by modeling a *conditional* distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{y})$, conditioned on some $\boldsymbol{y}$. In the case of MNIST, we could condition on the number we want to generate, e.g $p_{\boldsymbol{\theta}}(\boldsymbol{x}|y=3)$ for the number 3. For depth maps it is more complicated, as we want to generate a depth map of a given image, and we need 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 3.7b. Here, the conditioning is implemented by concatenation, which effectively gives the network information about what it should reconstruct.

The variational lower bound in (3.14) can be re-written (without individual samples $\boldsymbol{x}^{(i)}$ for notational simplicity) using the conditional distribution as

$$\begin{aligned}\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{y}) &\geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) \\ &= -\mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y}) \middle|\middle| p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{y})\right) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{y})\right], \end{aligned} \tag{3.23}$$

giving the SGVB estimator

$$\begin{aligned}\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y}) = &-\frac{1}{2}\left(\mathrm{tr}(\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y})) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y})^{\mathsf{T}}\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y}) - k - \log|\boldsymbol{\Sigma}_{\boldsymbol{z}}(\boldsymbol{x}, \boldsymbol{y})|\right) \\ &+ \left(-\frac{k}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}, \boldsymbol{y})| - \frac{1}{2}\left\|\boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}, \boldsymbol{y})\right\|^2_{\boldsymbol{\Sigma}_{\boldsymbol{x}}}\right). \end{aligned} \tag{3.24}$$

As seen from (3.24), the objective to be optimized for the CVAE is exactly the same as for the VAE in (3.22).

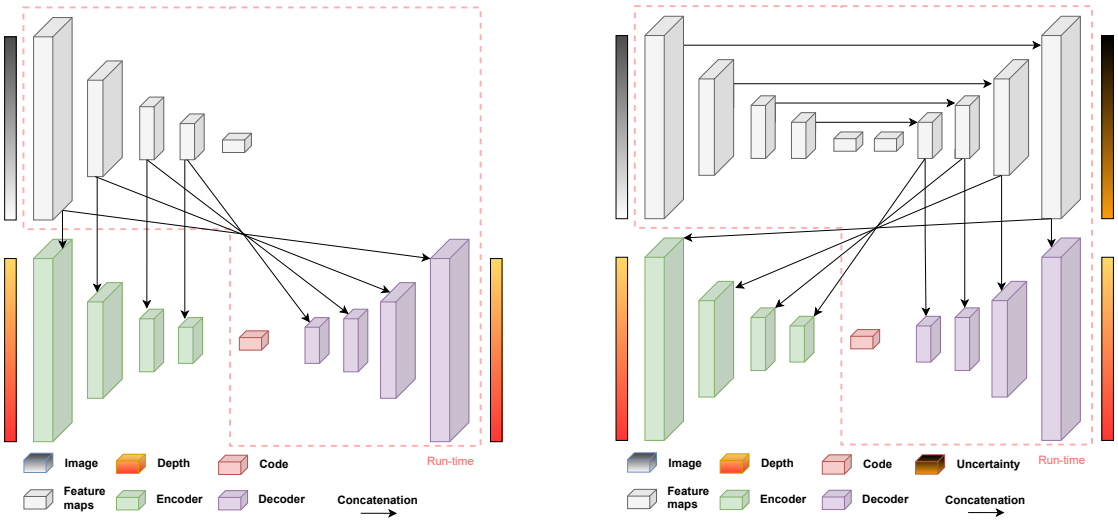
In practice, weighing the reconstruction error by the covariance of the prediction $\boldsymbol{\Sigma}_{\boldsymbol{x}}(\boldsymbol{z}, \boldsymbol{y})$ is strictly not necessary, but often beneficial. After all, the output of interest for the VAE and CVAE is the reconstruction itself (or rather the 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

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

which results in a mean squared error (MSE) loss for the reconstruction. This doesn't 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. This is identical to estimating the aleatoric uncertainty as in [23].

CVAEs for depth estimation

Although these models originate and have their main use within generation of data, recent works have attempted to adapt them to inference of data. Most notably, [8] 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 image, with far fewer parameters, and has two main benefits:



(a) *CVAE conditioned on deep features from a contracting CNN.*

(b) *CVAE conditioned on deep features from the expanding path of a U-Net [3], additionally estimating the uncertainty of the prediction.*

Figure 3.8: *CNN-based implementations of CVAEs for depth estimation from intensity images.*

1. The reduced parameter-space allows for joint optimization of pose and map in dense VSLAM and SfM.

2. The compressed representation is computationally less memory-demanding than its full counterpart, consuming less memory when storing the map of the environment.

The first point is the main contribution to the VSLAM community, as optimizing pose and map in a dense manner has previously not been possible due to computational real-time constraints. However, the accuracy in the predicted depth maps, and particularly how well the code and decoding captures the full structure, is a limitation of this method. Compared to state-of-the-art methods for monocular depth estimation [7], [24], neither of the CVAE-based methods [8], [17]–[19] 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 [8], 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 use a pre-trained classification CNN, extract the feature maps from it and concatenate them with the features in the encoder and decoder of the CVAE, as shown in Figure 3.8a. 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. As the learned features from a pre-trained network may not be optimal for depth estimation, the top stream may instead be trained together with the bottom stream.

As the SGVB estimator in (3.24) naturally incorporates uncertainty, we can estimate the uncertainty of the depth 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 by itself for further inferring information about the estimated depth, as it enables the possibility of choosing which estimates to use and 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 3.8b.

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 becomes infeasible (at least in real-time) due to the large number of parameters the dense map presents in the optimization. As shown in [8], 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.

4 | Multi-view depth estimation

As estimating depth from single images is inherently ill-posed, a natural question is how multiple images can be leveraged in deep learning to estimate depth. Since the targeted application is still inferring depth from a monocular setup, the multi-view methods cannot rely on stereo imaging, severely restricting the number of relevant methods. This chapter will thus only focus on temporal multi-view methods, in addition some adaptations using stereo configurations during training, but not during testing. As the methods share many of the underlying concepts described in Chapter 3, the discussion will be very brief here, only focusing on the new core ideas.

4.1 Temporal and static multi-view

Compared to the single-view methods in Chapter 3, which all need supervision by known depth, the multi-view methods can take advantage of other signals, like using direct photometric measurements and relative poses between images to infer depth. This is identical to the core concept in direct SfM or VSLAM, namely finding map and/or pose by minimizing photometric errors between images,

$$e_p(\boldsymbol{u}^b, z^b, \mathbf{T}_{ab}) = I_a(w(\boldsymbol{u}^b, z^b, \mathbf{T}_{ab})) - I_b(\boldsymbol{u}^b), \tag{4.1}$$

where I_a and I_b are two images with overlapping views, and $w(\cdot)$ is the warp function for transforming pixels $\boldsymbol{u}^b$ in I_b to the corresponding pixels $\boldsymbol{u}^a$ in I_a using the depth z and relative pose between the images $\mathbf{T}_{ab}$. The depth and relative pose are usually unknown, and are the variables we need to solve for, typically done by non-linear optimization. This does, however, assume that photometric and geometric distortions are not present in the images, such that the photometric error should be zero for corresponding pixels.

In [9], the depth is estimated by an autoencoder that aims to minimize the photometric error between consecutive images in a stereo setup, using temporal and/or static views. The relative pose between the stereo image pairs are naturally known, but not for temporally adjacent frames, which is why the pose also must be estimated in this case, here by a separate CNN. The key contribution is that this training setup does not require ground truth depth, which in many cases is difficult to acquire, and the training is therefore self-supervised. At run-time, the network has learned to predict depth from single images, and the method therefore works for monocular setups as well. There are various improvements and implementation details that will not be expanded upon here, like using SSIM[25] for photometric errors and edge-aware smoothness (3.5) as a geometric prior, and further details are left to the reader.

Sharing many similarities with the previously described method, [10] estimates the *disparity*

maps of two stereo images instead of the depth directly, which relates to depth by

$$\text{disp} = \frac{b \cdot f_x}{z}, \tag{4.2}$$

where b is the baseline between the two views, f_x is the focal length along the x-axis of the calibrated cameras and z is the depth. The disparity maps for both the left and right images are predicted from an autoencoder using only the left stereo image, and the training signal is a combination of multiple loss functions using the reconstructed right stereo image from the disparity maps. Additionally, a subsequent autoencoder is used to refine the predicted disparity maps, calculating the residual between the predicted disparity and actual disparity. As the ground truth disparity maps can be calculated from the stereo image pairs, the method is also self-supervised. Interestingly, a supervised signal is also used to improve the predictions, where estimated sparse disparities from Stereo DSO[26] are used in aiding the prediction of the full disparity map. Additional loss functions are left-right consistency loss and regularizers for local smoothness and occlusions, which will not be expanded on here.

A useful property of [10] is that the network learns to predict the corresponding stereo image from a monocular image by the disparity maps. The input to the network can be interpreted as e.g the left image, and it will be able to reconstruct the corresponding right image in a stereo setup using the predicted disparities. The baseline between the images will be the same baseline as used when training, and is therefore dependent on the stereo setup in the training set. This *virtual stereo* can be further used to improve the accuracy of odometry or SLAM methods based on e.g the self-supervised methods described here, by incorporating a photometric loss based on the virtual stereo image pairs.

A question one may ask is how multiple views can be incorporated into the single-view methods in Chapter 3, using both ground truth depth maps and signals available with multiple views to supervise the training. Intuitively, this should give better performance than using individual images alone, since estimating depth inherently requires several views. The author is currently not aware of any methods that fuse the two, and research into how this can be done is left as possible future work.

5 | Method

The two previous chapters have served as a review of some of the state-of-the-art methods for estimating depth from images, with the purpose of understanding and deciding which methods are most applicable to the problem at hand. This chapter aims to first weigh the different methods against each other, with respect to underwater mapping using monocular images, before finally deciding on a method. Finally, implementation details of the chosen method is described.

5.1 Choice of method

Before delving into the methods, some words should be said about the data they will be applied to. The Eelume underwater snake robot provides monocular images in a harsh underwater environment, with examples of possible test data shown in Figure 5.1. The data collected at the present time does not contain any measurements which can be used as ground truth, mainly due to the difficulty of acquiring this underwater, and the images have highly varying lighting conditions and photometric distortions, which is a natural product of the underwater environment. Two important considerations should be made from this alone:

1. Methods requiring ground truth depth cannot be trained on the data from the robot.
2. Methods relying on photometric consistency, like using photometric error, will most likely fail due to the photometric distortions.

As the methods in Chapter 3 require ground truth depth, and the methods in Chapter 4 assume photometric consistency, it appears none of the methods are suited. This should, however, not be viewed as a limitation of the reviewed methods, but more so evidence of how challenging the targeted application is. Since training on this data is not possible without modifying the reviewed methods heavily, this work will use other data that hopefully translate well to the underwater environment, later described in Chapter 6.

The multi-view methods discussed in Chapter 4 are the only methods directly applicable to the underwater data, and are from this consideration alone advantageous to the methods in

Figure 5.1: *Examples of images captured by the Eelume snake robot.*

Chapter 3. This is, however, not the case for the data used in this work, which has known ground truth, but also has a photorealistic environment with photometric distortions. These methods will therefore ultimately fail without complex modifications, and are therefore not considered further in this work. Although they are not considered here, solving the issues related to photometric consistency is still an important area, and is therefore left as future work.

The Transformer of [7] is the current state-of-the-art for depth estimation, and may therefore seem like a good choice for further investigation. The authors, however, report that it requires large amounts of training data to achieve state-of-the-art performance, and performs sub-par on smaller datasets. The solution is to train on a large, similar dataset, or use a pre-trained network, before using transfer learning on the targeted data. As there isn't any large dataset of similar data to the ones considered here, this may give poor results, and the Transformer is therefore not regarded as suitable for this work.

The plain autoencoders are perhaps the simplest models, but still perform relatively well compared to the Transformers. The details of Chapter 3.1 are lacking with regard to the modifications and complexity that can be done with an autoencoder, and e.g [27] use a plain autoencoder with a more complex pyramid-scheme in the decoder to achieve near state-of-the-art performance. They are therefore good candidates for further investigation.

The variational counterpart to the autoencoder, as thoroughly presented in Chapter 3.3.1, seems to perform worse than the plain autoencoders, mainly due to the regularization that the latent code presents. However, the latent code itself exhibits useful properties for further refinement of the dense geometry, as the reduced parameter-space is more easily optimized over in e.g photometric bundle adjustment. It is difficult to say exactly how well this refinement performs compared to predictions from other methods, but due to the useful properties for further dense 3D reconstruction and VSLAM, the variational autoencoder is chosen for further investigation, and specifically the CVAE for depth estimation. As this model consists of a plain autoencoder in its top stream, it is also easy to test the plain autoencoder for depth estimation using this architecture.

5.2 Implementation

A CVAE was implemented[1] in PyTorch[28], adapting and expanding the deep learning framework provided by [29] to depth estimation instead of detection. The network, termed Depth-CVAE, is shown in Figure 5.2, with additional details given in Table B.1 and Table B.2 in Appendix B. Much inspired by [8], the top stream is a U-Net that predicts the uncertainty map for the predicted depth, and the bottom stream is the CVAE conditioned on the deep features from the U-Net, which ultimately predicts the depth of each pixel in the image.

5.2.1 Loss function

There are some exceptions in the implementation from what is outlined in Chapter 3.3.1. The traditional noise model is a Gaussian, but the Depth-CVAE uses a Laplace distribution for modeling the noise,

$$p(x) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right), \tag{5.1}$$

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

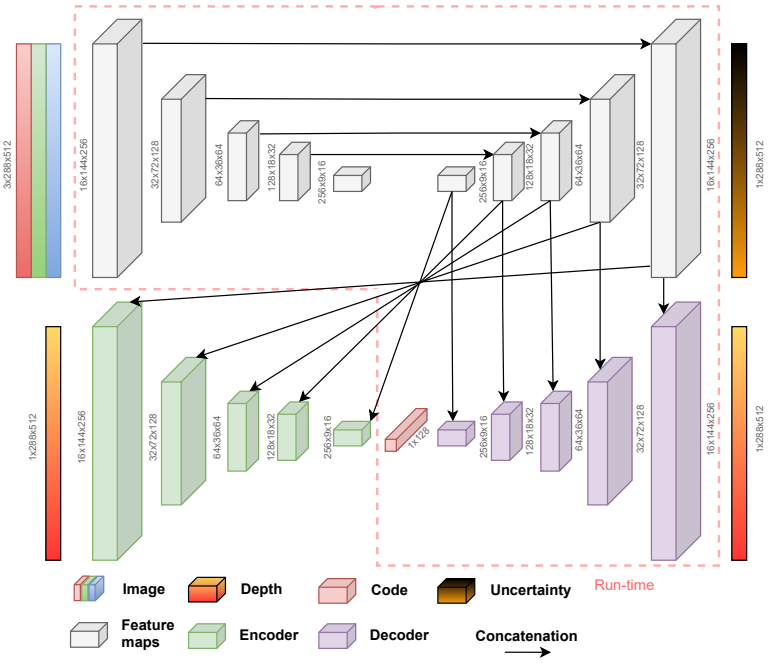


Figure 5.2: *Graphical structure of the CNN-based CVAE (Depth-CVAE) used in the experiments, with the example of RGB images of size* 288×512 *as input.*

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 [4].

Additionally, the code layer normally has a prediction of $\boldsymbol{\Sigma}_{\boldsymbol{z}}$, but is in the Depth-CVAE predicted as $\log \boldsymbol{\Sigma}_{\boldsymbol{z}}$ instead. This is to increase the numerical stability, as the KL divergence in (3.24) involves taking the log of $\boldsymbol{\Sigma}_{\boldsymbol{z}}$, which is unstable for values close to zero.

A common problem when training VAEs using the objective in (3.22) is that the KL divergence is quickly driven towards zero[30]–[32]. This intuitively doesn't sound like a problem, as we after all are aiming to minimize the objective, but it heavily regularizes the model from the beginning. The result is a latent representation 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 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

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}) = \mathcal{L}_{\text{recon}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}) + \beta \mathcal{L}_{\text{KL}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}), \tag{5.2}$$

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 $\beta = 1$. A more advanced schedule can be seen in [32].

5.2.2 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

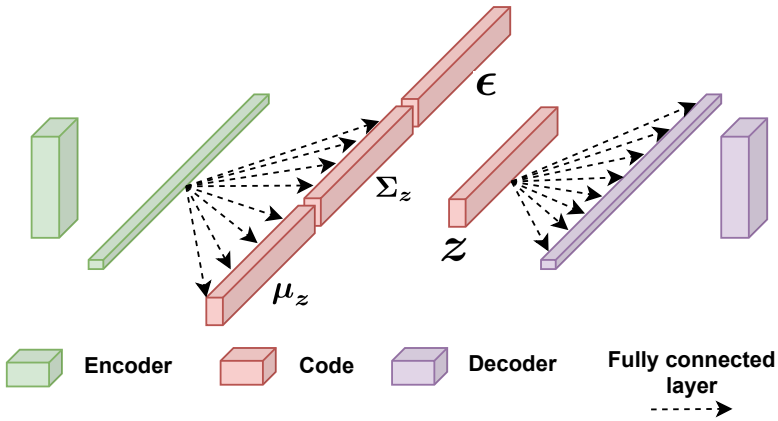


Figure 5.3: *Implementation of the code using neural networks. The final layer of the encoder is flattened and forwarded through two separate fully connected layers that each make up the expectation and covariance of the code distribution. The code $\boldsymbol{z}$ is then sampled using the reparameterization* (3.21)*, and forwarded through one fully connected layer to generate the first decoder layer.*

connected layers are used for each of them. A possible implementation of the code block is shown in Figure 5.3. The covariance matrix is assumed to be diagonal, such that each element of $\boldsymbol{\Sigma}_{\boldsymbol{z}}$ in Figure 5.3 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 $\boldsymbol{z} = \mathbf{0}$.

6 | Results and discussion

This chapter presents the experimental setup and results for the implemented CVAE for monocular depth estimation. The datasets used for the experiments are first presented, before results and details from experiments are finally shown, demonstrating the performance of the CVAE.

6.1 Datasets

Two datasets were used in the evaluation of the CVAE: the SceneNet RGB-D [33] dataset and the VAROS Synthetic Underwater [34] dataset. Both provide photorealistic synthetic RGB images generated in artificial environments, with corresponding precise ground truth depth images. The benefit of using synthetic data is that the much needed ground truth depth is available, which otherwise is usually obtained with depth-measuring sensors such as RGB-D cameras or laser scanners, preferably highly accurate, but these fail in the case of underwater environments. The extent of realism in the synthetic scene is however a possible bottleneck, especially how well the photorealistic images translate to real scenes. The question remains whether a network trained on synthetic images will perform sufficiently on real underwater data.

6.1.1 The SceneNet RGB-D dataset

SceneNet RGB-D is a dataset composed of photorealistic renderings of different indoor environments, with a total of 5M rendered RGB-D images from 15K trajectories of different scenes. The large amount of available data makes this dataset suitable for deep learning tasks, such as semantic segmentation, object detection and, most importantly, depth estimation. Some examples from the dataset can be seen in Figure 6.1. As seen from the rightmost images in Figure 6.1, the

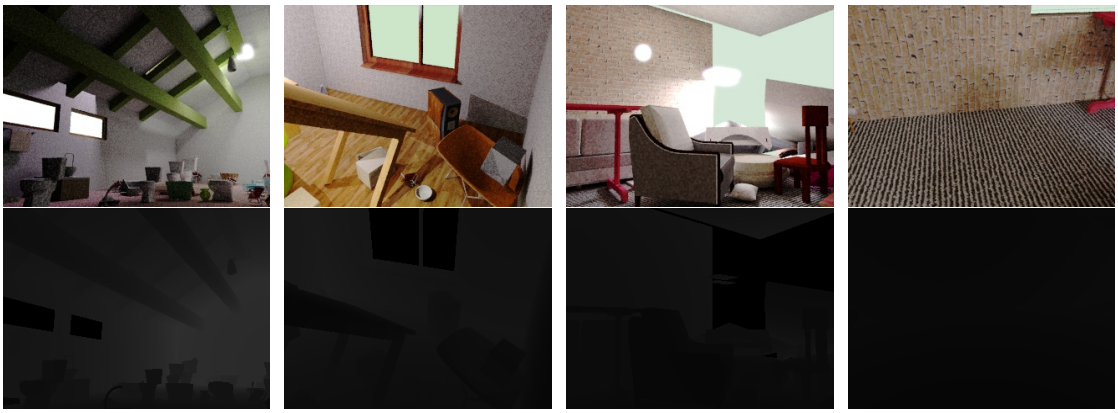
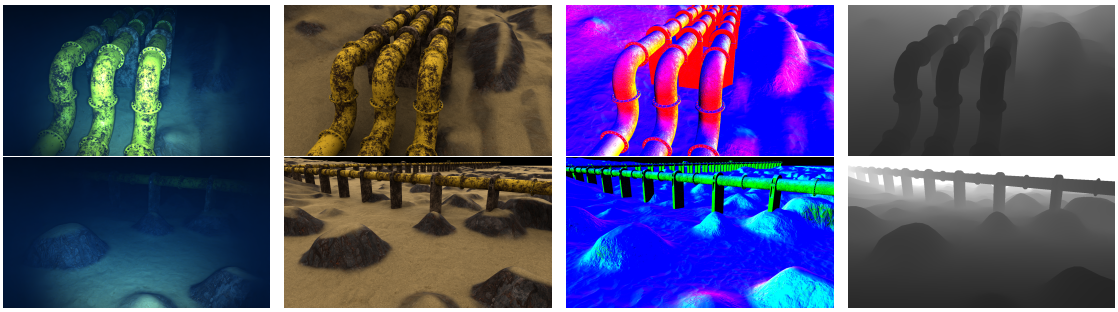


Figure 6.1: *Examples of the SceneNet RGB-D images (top) and corresponding depth (bottom).*



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

Figure 6.2: *Examples of the VAROS dataset.*

dataset contains some images that are poorly suited for depth estimation, and training a deep network on many of these images will negatively affect its performance.

The dataset was primarily used as a means of verifying the working condition of the implemented CVAE, as the dataset had previously been successfully used in [8]. The scenes do not represent the targeted underwater environment, however, and will therefore not be used in future work, but it gave valuable insight during initial testing. A possible further use could be to use it as a pre-training dataset, as learned features and structure may still be transferable to other data, but this was not tested here.

6.1.2 The VAROS Synthetic Underwater dataset

VAROS is a dataset composed of photorealistic renderings of an underwater environment, with a current total of 4715 rendered RGB images, with corresponding depth maps, surface normal maps, and uniformly lighted images with no water. Two examples of the scene and its corresponding data can be seen in Figure 6.2.

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

6.2 Experiments

The Depth-CVAE was trained and tested on both SceneNet RGB-D and VAROS, but without substantial hyperparameter-tuning, architecture experimentation, data augmentation, pre-training or other possible ways of optimizing the performance. The extent of this work is mainly to test and verify the working degree of a CVAE on underwater images, and further optimization of its performance is left as future work. Qualitative results are therefore presented in this work, with more quantitative results reserved for later.

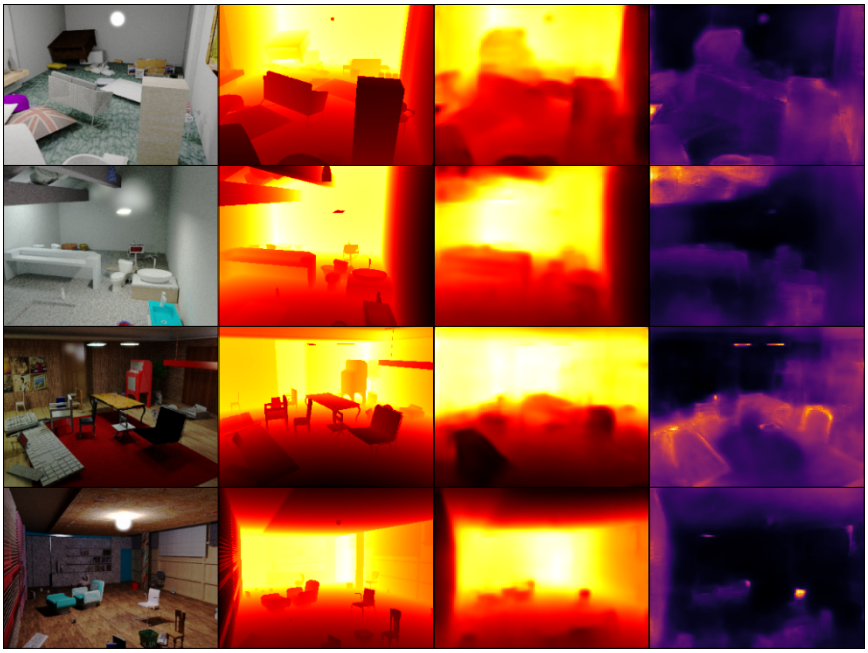


Figure 6.3: *Examples from the test set of SceneNet RGB-D using the Depth-CVAE. For ground truth depth, predicted depth and uncertainty, brighter areas indicate higher values.*

6.2.1 Experimental setup

All experiments were conducted on a NVIDIA GeForce GTX 1060 6GB with hyperparameters proposed in [8], using the Adam optimizer with learning rate $\alpha = 0.0001$ and $\beta_1 = 0.9$, $\beta_2 = 0.999$, training for 10 epochs using a batch size of 32. A KL annealing schedule was used during training, gradually increasing the weight on the KL divergence from 0 to 1 by a sigmoid function after 3 epochs. The RGB images were normalized using the calculated mean and standard deviation for each dataset, and the depth values were transformed to the range [0,1] by the proximity parameterization proposed in [8]. The dimension of the code was set to 128.

6.2.2 SceneNet RGB-D

SceneNet RGB-D provides 17 training set splits, each composed of around 300K images, but only the first was used for training due to memory constraints on the computer. The official test set was not available at the time of testing, so another training set split was used for validation instead. Results from testing on the test set can be seen in Figure 6.3, with additional results shown in Appendix C.

As seen in Figure 6.3, the model is successful in estimating the main structure of the scene, but struggles with finer details. This is not uncommon for VAEs[35], which have the disadvantage of producing blurry or smoothed results compared to e.g GANs. The advantage of the compact representation of the images in the code, however, still remains, which could further be used to refine the estimated depth as in [8], [17], [19]. The uncertainty map can also be seen to provide

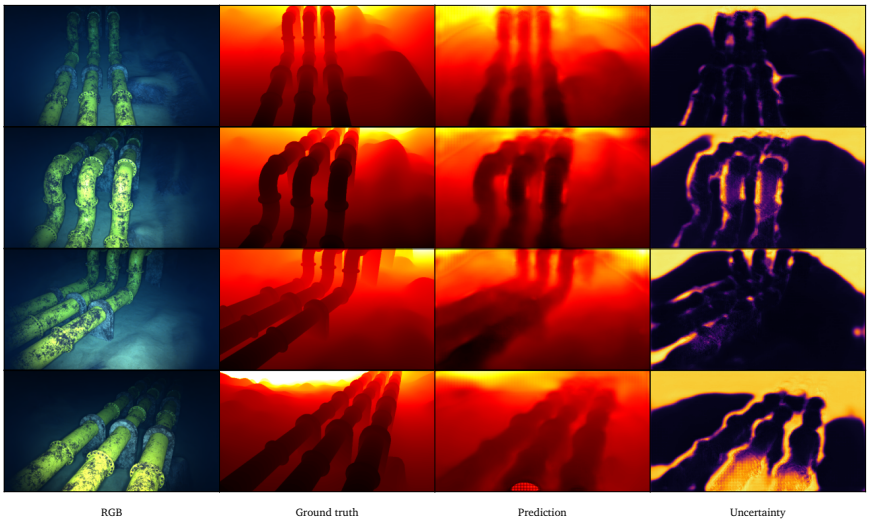


Figure 6.4: *Examples from the test set of VAROS using the Depth-CVAE. For ground truth depth, predicted depth and uncertainty, brighter areas indicate higher values.*

useful information in which areas in the prediction to trust, and which to not trust. In e.g the second row of Figure 6.3, the network struggles to predict the beams in the upper left corner, but also allocates higher uncertainty in that region. This gives us the opportunity of discarding possibly poor predictions, and e.g keeping only the ones within a certain threshold.

6.2.3 VAROS

The test set for VAROS was difficult to choose, due to its limited size and most notably the sequential recording of data. Random images could therefore not be removed and used in the test set, as they would have very similar images in the training set, giving a highly biased indication of the models performance. A larger *sequence* of data was therefore removed from the training set 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. Specifically, all images that viewed the pipes, as seen in Figure 6.2, from the right side were removed. This resulted in a training set of 4265 images, and a test set of 450 images. Results from the test set can be seen in Figure 6.4, with additional results shown in Appendix C.

Similar to the results from SceneNet RGB-D, the Depth-CVAE succeeds in capturing the main structure of the scene, but again struggles with the finer details. Perhaps even less so than for SceneNet RGB-D, but this is probably due to fewer finer details in VAROS, and because the network is, still, to a large degree specialized on the few scenes it sees. It would be more interesting to see how it predicts the structure of an object is has not previously seen, which would give a better indication of the reliability of the model. As seen from the uncertainty in Figure 6.4, the network allocated higher uncertainty around the upper edges of the scene, which corresponds well to the RGB images as this region is unobservable. The last example also shows an artifact in the prediction, indicating that there may be some instability in the predictions, but the corresponding uncertainty is also high in that region.

To further test the capabilities of the model, the images in Figure 6.4 were vertically flipped

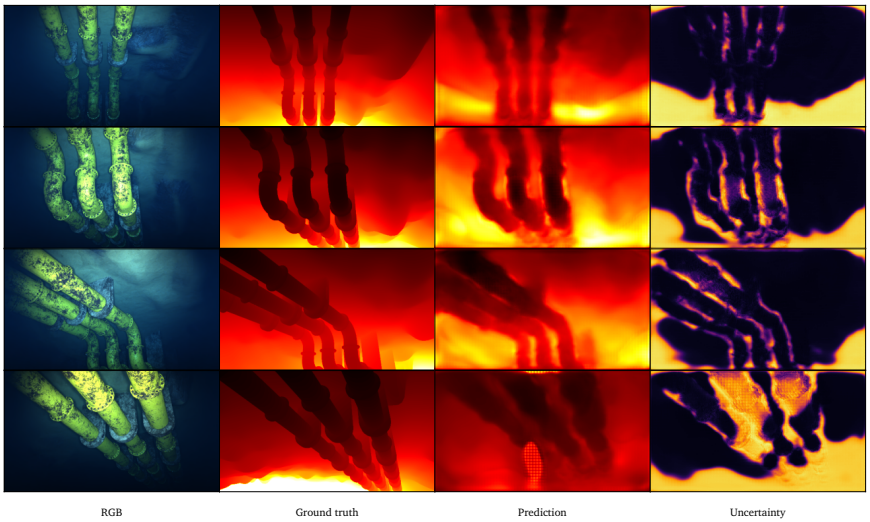


Figure 6.5: *Examples from the test set of VAROS with vertically flipped images.*

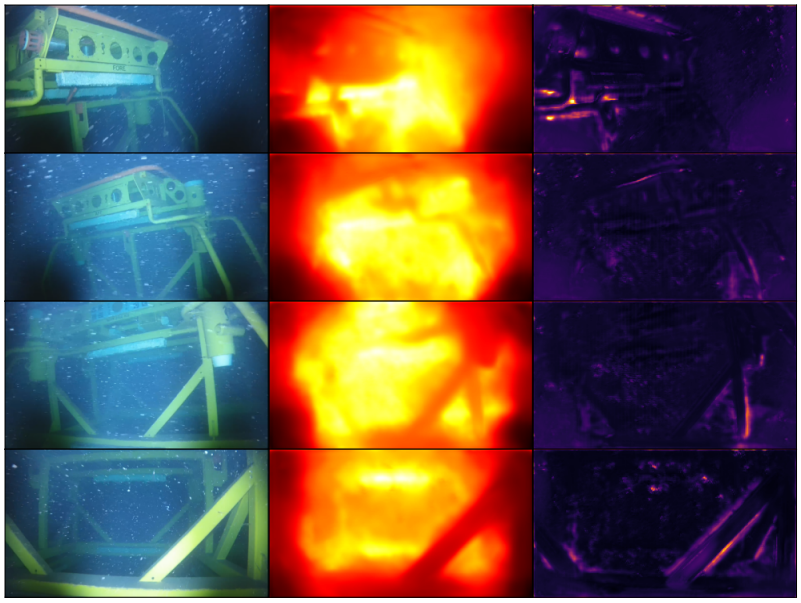
before giving them to the network. This should present the model with an even further diverse scene, which is also realistic for the robot, as it may rotate its body upside down. The results are shown in Figure 6.5, producing similar results to Figure 6.4, but with slightly higher uncertainty. The artifact, oddly enough, still remains on the lower part of the prediction, in regions with low uncertainty, and should therefore be investigated in future work.

6.2.4 Eelume footage

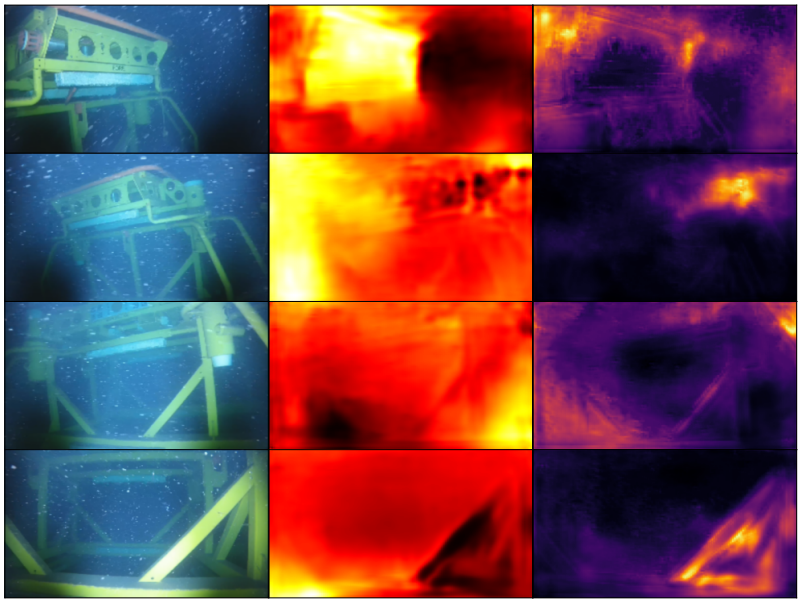
The final question, and perhaps the most interesting, is how well the trained model transfers its knowledge to real underwater scenes. Estimating the depth in real underwater scenes is, after all, the end goal of this work. Results using images captured by the Eelume snake robot is shown in Figure 6.6, using a model trained on either VAROS or SceneNet RGB-D. As seen, the model struggles heavily in capturing the geometry of the images.

The model trained on VAROS predicts some of the geometry and structure correctly, but even for these areas it is difficult to say how correct the predictions actually are. The results are at first sight disappointing, but not unexpected. Since the VAROS dataset is so small, and the variety of the scenes so scarce, it would be surprising if the model correctly predicted the structure of an object it has previously not seen. The only man-made object it has seen is the pipes in Figure 6.4, which alone is not enough to correctly predict the more complex object in Figure 6.6. For this reason, a model trained on SceneNet RGB-D was also evaluated, as this dataset contains both more data and more objects. The results in Figure 6.6b are, however, even worse. This could be explained by the object in the scene, which is different from those in SceneNet RGB-D, or the underwater images, which have very different image intensities than SceneNet RGB-D.

A question is also how well the synthetic data of VAROS translates to real underwater images, and if this is a deciding factor. It is difficult to reason about this, as the limited size of the dataset is likely a larger contributor to the poor performance. The model has also not been subject to regularization, like data augmentation, which may improve the results on the Eelume footage.



(a) *Results using a network trained on VAROS.*



(b) *Results using a network trained on SceneNet RGB-D.*

Figure 6.6: *Results on footage from the Eelume snake robot.*

7 | Conclusion

The goal of this report was to investigate depth estimation methods based on deep learning for monocular images, and to propose, implement and test a suitable method on data from an underwater environment.

To accomplish this, several state-of-the-art methods for depth estimation using monocular images were reviewed. Deep learning architectures based on autoencoders, Transformers, variational autoencoders and multi-view photometric consistency were considered, with emphasis on their theoretical inner workings. This was done in order to propose a suitable method, based on one or multiple of the reviewed methods, for inferring depth in underwater environments.

A variational autoencoder was implemented, conditioned on the deep features of the images it predicts depth from. The model was trained and tested on two datasets: the SceneNet RGB-D dataset for verification and comparison of its performance to a state-of-the-art variational autoencoder, and the VAROS Synthetic Underwater dataset for final testing of its performance in an underwater environment. Qualitative results show that the model performs well on both datasets, but the limited size of VAROS makes it difficult to reason on its generalization to more diverse underwater scenes. Experiments on real underwater footage from the Eelume robot indicate that the model has much to learn, or perhaps that the synthetic data doesn't translate well to the real scenes it aims to model.

7.1 Further work

There are numerous possible ways of improving the models performance on depth estimation. Improvements of the currently implemented model can be considered by itself, but there are also more substantial changes to the overall depth estimation pipeline to be considered. In the following, some ideas and potential improvements are listed for further work.

The implemented model was not thoroughly explored:

- The architecture of the model may not be optimal for the problem, and further research into how this can be changed should be done. E.g using residual connections, allowing a deeper encoder and decoder, may improve performance.
- The U-Net predicts the uncertainty of the predictions, but a possible improvement may be to predict both the uncertainty and depth, as in [24]. This way, the decoder of the U-Net has feature maps that are more directly linked to deep features of depth, and the CVAE is perhaps better conditioned to predict depth as well.
- Additional information can be given to the Depth-CVAE, e.g a sparse depth map as in [17]. In [19], the reprojection error map is additionally given, resulting in better predictions.

- More complex loss functions can be used to give stronger geometric priors, for instance using the edge-aware smoothness regularizer (3.5).
- Using pre-trained encoders may expose the model to a larger variety of data, and thus improve performance. This should also decrease the amount of training time required for the model to learn optimal parameters. An interesting experiment could also be to pre-train on e.g SceneNet RGB-D to learn geometric structures, and fine tune on VAROS.
- Data augmentation was not used to allow the creation of a representable test set, but should be done to increase generalization. For instance, since an AUV may rotate about all axes in three-dimensional space, both horizontal and vertical flipping with rotations are relevant augmentations.
- Hyperparameters always leave room for improvement, and since the model trained in this work did not undergo considerable tuning, this should be further looked into.

As discussed in Chapter 6, the limitations of VAROS impose some difficulties for deep learning models:

- More data should be collected to expand the number of scenes the model can train on. This shouldn't only be sequences from a robot trajectory, but more pseudo-random scenes. A possible solution is to sample images from pseudo-random poses in the underwater environment, which can substantially increase the number of images. These must, however, be valid and realistic, which may be difficult to implement.
- The underwater environment should be expanded to include more diverse scenes, and should e.g contain more man-made objects that are likely to be seen in real footage.

A big disadvantage of the proposed model is that it can't train on real footage without having ground truth depth, which is difficult to obtain underwater. The CVAE is also regularized towards its own prior, leading to poorer predictions (but with the upside of a smooth and optimizable latent space). Some ways to tackle these problems are:

- Propose a way to solve the photometric inconsistency in underwater images, making multi-view methods applicable. This allows the use of other deep learning methods, but also makes it possible to refine the predictions through multi-view optimization based on photometric errors, realizing the strength of the proposed CVAE.
- Propose other metrics that allow multi-view optimization, which are robust to the visually degraded images from underwater environments.
- Implement and test other deep learning methods for depth estimation in the underwater environment.

Bibliography

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

[2] 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].

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

[4] 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].

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

[6] 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].

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

[8] 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].

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

[10] 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].

[11] 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.

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

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, 'Attention is all you need,' 2017. arXiv: 1706.03762 [cs.CL].

[14] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit and N. Houlsby, 'An image is worth 16x16 words: Transformers for image recognition at scale,' 2021. arXiv: 2010.11929 [cs.CV].

[15] J. Sivic and A. Zisserman, 'Efficient visual search of videos cast as text retrieval,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 4, pp. 591–606, 2009. DOI: 10.1109/TPAMI.2008.111.

[16] A. C. Kumar, S. M. Bhandarkar and M. Prasad, 'Monocular depth prediction using generative adversarial networks,' *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 413–4138, 2018. DOI: 10.1109/CVPRW.2018.00068.

[17] 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].

[18] 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.

[19] 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].

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

[21] 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.

[22] 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.

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

[24] 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].

[25] Z. Wang, A. Bovik, H. Sheikh and E. Simoncelli, 'Image quality assessment: From error visibility to structural similarity,' *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. DOI: 10.1109/TIP.2003.819861.

[26] R. Wang, M. Schwörer and D. Cremers, 'Stereo dso: Large-scale direct sparse visual odometry with stereo cameras,' 2017. arXiv: 1708.07878 [cs.CV].

[27] 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.

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

[31] 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].

[32] 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].

[33] 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.

[34] 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.

[35] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky and A. Courville, 'Adversarially learned inference,' 2017. arXiv: 1606.00704 [stat.ML].

[36] 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

We want 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 (3.13) 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 (3.10) on $\log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, we get

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

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

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

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 we have used identities from [36] and the cyclic property of the trace. 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	Downsample	-	-	-	$3 \times 144 \times 256$
		DoubleConv 3×3	ReLU	BN	-	$16 \times 144 \times 256$
	2	Downsample	-	-	-	$16 \times 72 \times 128$
		DoubleConv 3×3	ReLU	BN	-	$32 \times 72 \times 128$
	3	Downsample	-	-	-	$32 \times 36 \times 64$
		DoubleConv 3×3	ReLU	BN	-	$64 \times 36 \times 64$
	4	Downsample	-	-	-	$64 \times 18 \times 32$
		DoubleConv 3×3	ReLU	BN	-	$128 \times 18 \times 32$
	5	Downsample	-	-	-	$128 \times 9 \times 16$
		DoubleConv 3×3	ReLU	BN	-	$256 \times 9 \times 16$
Decoder (D)	0	DoubleConv	ReLU	BN	-	$256 \times 9 \times 16$
	1	Upsample	-	-	-	$256 \times 18 \times 32$
		Conv 3×3	-	-	-	$128 \times 18 \times 32$
		Concatenate	-	-	E4	$256 \times 18 \times 32$
		DoubleConv 3×3	ReLU	BN	-	$128 \times 18 \times 32$
	2	Upsample	-	-	-	$128 \times 36 \times 64$
		Conv 3×3	-	-	-	$64 \times 36 \times 64$
		Concatenate	-	-	E3	$128 \times 36 \times 64$
		DoubleConv 3×3	ReLU	BN	-	$64 \times 36 \times 64$
	3	Upsample	-	-	-	$64 \times 72 \times 128$
		Conv 3×3	-	-	-	$32 \times 72 \times 128$
		Concatenate	-	-	E2	$64 \times 72 \times 128$
		DoubleConv 3×3	ReLU	BN	-	$32 \times 72 \times 128$
	4	Upsample	-	-	-	$32 \times 144 \times 256$
		Conv 3×3	-	-	-	$16 \times 144 \times 256$
		Concatenate	-	-	E1	$32 \times 144 \times 256$
		DoubleConv 3×3	ReLU	BN	-	$16 \times 144 \times 256$
	5	ConvTranspose 2×2	Sigmoid	BN	-	$1 \times 288 \times 512$

Table B.1: *Detailed network architecture for the U-Net stream of the Depth-CVAE. Downsample layers use 2×2 max-pooling, BN is batch normalization, DoubleConv $n \times n$ applies a $n \times n$ filter two times, each with activation and normalization given in the table, Upsample layers use bilinear interpolation and ConvTranspose $n \times n$ uses a $n \times n$ transposed convolution for upsampling.*

B.2 CVAE

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

Table B.2: *Detailed network architecture for the CVAE stream of the Depth-CVAE. Downsample layers use 2×2 max-pooling, DoubleConv $n \times n$ applies a $n \times n$ filter two times, each with activation and normalization given in the table, Upsample layers use bilinear interpolation and ConvTranspose $n \times n$ uses a $n \times n$ transposed convolution for upsampling. The linear layers in the code are separate,* ***not*** *subsequent, and each predict $\boldsymbol{\mu}$ or $\log \boldsymbol{\Sigma}$.*

C | Additional results

C.1 SceneNet RGB-D

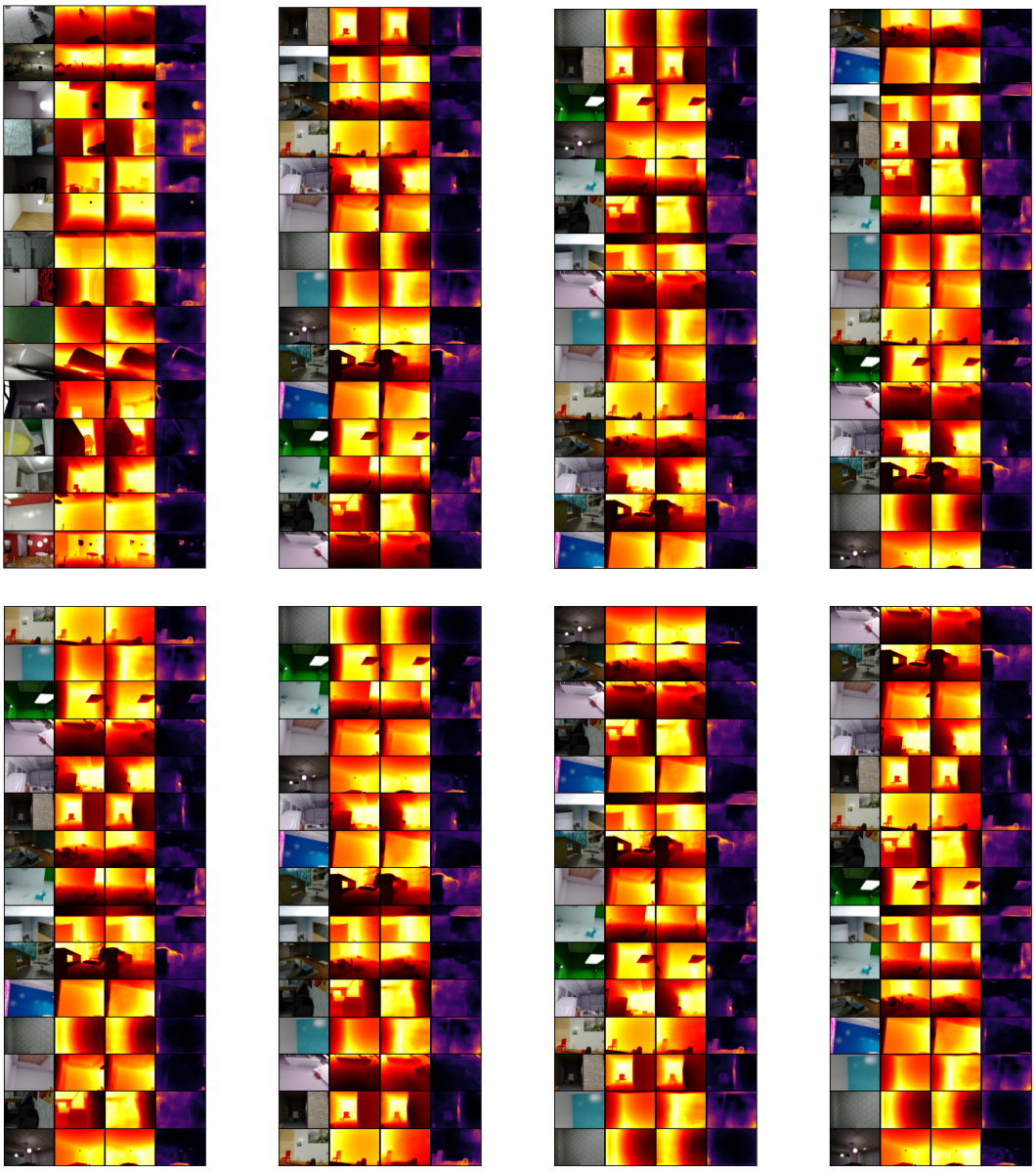


Figure C.1: *Additional results from the SceneNet RGB-D test set, with randomly drawn samples. Each figure displays (from left to right): RGB, ground truth depth, predicted depth and uncertainty. Zoom in to better view the results.*

C.2 VAROS

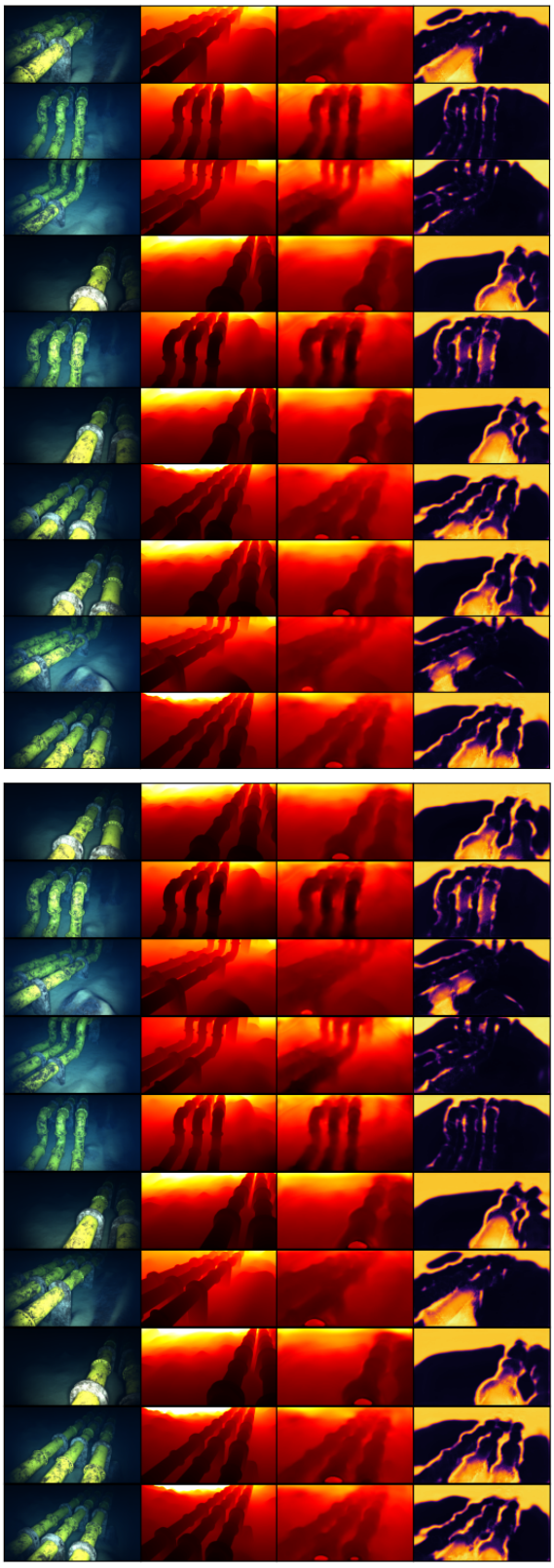


Figure C.2: *Additional results from the VAROS test set, with randomly drawn samples. Each figure displays (from left to right): RGB, ground truth depth, predicted depth and uncertainty. Zoom in to better view the results.*