Uncertainty in Self-supervised Depth Estimation Using Multi-scale Decoders

Mayank Mali

CMU-CS-22-124

August 2022

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213

Thesis Committee:
Dr. Jean Oh (advisor)
Dr. Ji Zhang

Submitted in partial fulfillment of the Masters degree in Computer Science

Keywords: Machine Learning, Depth Estimation, Self-Supervised, Uncertainty, Uncertainty Estimation, View Synthesis, Sparsification, Student-Teacher Framework, U-Net, Decoder, Multi-Scale, ResNet

For my family, friends, and faculty that offered endless support and feedback.

Abstract

Depth estimation is an image translation problem that predicts depth maps for a given camera image and has fostered research in various applications including self-driving vehicles. Self-supervised depth estimation methods are of particular interest since ground truth LIDAR depth is expensive to acquire and instead use view synthesis as weaker supervision. Generally, the produced depth maps to date are only point estimates of an underlying depth distribution due to randomness in model training, resulting in noisy depth estimates that can propagate errors and lead to inaccurate or fatal decisions in real-world applications. Recent interest has been sparked in reducing such noise by modeling the uncertainty of depth estimates. Empirical uncertainty strategies seek to predict uncertainty via statistical methods by treating independent models as black box predictors. Of particular interest are predictive strategies that seek to learn the inherent uncertainty of a depth model. For example, student-teacher frameworks train one network to learn the depth output distribution of another. Such methods are desirable due to the advantage of requiring fewer training and space resources compared to other empirical methods. In this work, we study self-supervised depth models with a U-Net architecture that outputs depths at multiple scales. In particular, we explore a novel predictive uncertainty model that only has access to these scales and the U-Net bottleneck feature. We evaluate and discuss the novel method alongside other uncertainty strategies on the KITTI dataset.

Acknowledgments

Very special thanks to Dr. Soonmin Hwang for investing generous time in mentoring, teaching, and encouragement, extending the limits of my self-confidence, and accelerating my understanding of depth estimation. Also special thanks to my advisor Dr. Jean Oh for much-needed guidance, direction, support, and feedback. Thanks to Dr. Ji Zhang for giving time to be part of the thesis committee for the presentation and thesis review.

I also extend my gratitude to all my peers in the Bot Intelligence Group (BIG) at CMU, especially Ingrid Navarro, for their valuable time and effort in providing feedback for this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Monodepth: self-supervised monocular depth estimation

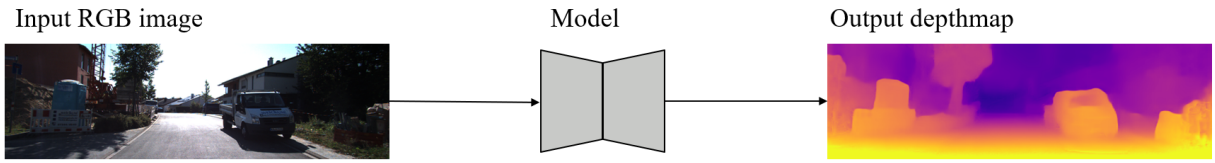


Figure 1.1: **Monocular Depth Estimation,** (Monodepth) is the problem of predicting pixel-wise depth maps given a single camera image.

The problem of monocular depth estimation (monodepth) is to predict depth maps[1] from a given camera image as shown in Figure 1.1. The underlying premise is that single images of an indoor/outdoor 3D scene contain various depth cues (i.e. object spatial arrangement, textures) that encode the distance from the camera i.e. depth. Furthermore, if camera images are collected via video or as stereo image pairs, then motion parallax between frames or stereo parallax respectively can give further depth cues.

The significance of this challenge is that pixel-wise depth maps are used to reconstruct and understand the 3D environment at the time the camera image was taken. Knowing the distance to an object and the geometry of the camera, allows a geometric approach to reconstructing the 3D scene. This task is particularly interesting for self-driving car applications [8].

However, ground truth depth points usually from LIDAR (Light Detection and Ranging) hardware are expensive to acquire. To address this problem, many works use view synthesis which is a geometry-based supervision. Sometimes called warping or reconstruction, this pipeline takes a camera image of a 3D scene from one viewpoint (i.e. camera pose) and predicts the image of the same scene for another viewpoint. For example, the KITTI dataset [4] offers stereo camera image pairs as the two views. Another option is to collect monocular video, and

[1]Depth maps from Convolutional Neural Networks (CNNs) usually output *inverse* depths, which are then flipped and clipped to a depth range. We use 0.1m to 80m as the minimum and maximum depth ranges.

use different frames of time as the viewpoints [16]. The latter technique is called SfM (Structure from Motion) and involves using visual odometry to understand the transformation between viewpoints.

1.2 Uncertainty for self-supervised depth estimates

Depth maps produced by monodepth models are point estimates, which have no basis on whether they should be trusted [12]. The sources of noise in the depth map estimates come from noise from the dataset (aleatoric) and model–inherent (epistemic) noises (e.g. from the randomness in training/initializations). Given a fixed camera image, there is an underlying *pixel-wise* depth output distribution that depends on the randomness in model training.

Uncertainty is then defined as the variation for depth prediction estimates along the output depth distribution. The goal of the uncertainty task is to predict the variation in the depth estimate, given the model and the camera image. The motivation for predicting uncertainty is that points far from the mean are more likely to have higher depth errors in evaluation. Therefore, we could improve the quality of depth estimates by predicting and then filtering the points with the highest uncertainty. It is important to note that Predicting uncertainty for self-supervised depth models is relatively new [11], whereas other works have predicted uncertainty for optical flow [10].

Uncertainty estimation hypothesis. If predicted uncertainty sufficiently encodes errors, we can improve depth accuracy by filtering the highest-uncertainty points. The importance of understanding this problem is that in self-driving vehicles that rely on depth map point estimates, high uncertainty in predictions can lead to fatal decisions. Furthermore, in robotics applications, keeping the most certain points for successive downstream tasks is the desired approach.

Chapter 2

Related works

2.1 Self-supervised monodepth

Due to the high cost of LIDAR ground truth depths and ease of capturing many images via video, much attention has been paid to self-supervised methods that use warping, or view synthesis [1, 3, 5, 6, 7, 8, 9, 13, 15, 16], a geometric technique that can supervise depth by transforming pixels in one camera image to another view. The final result is a synthesized camera image in another view/pose.

During training, a pair of images I_t, I_c from two different camera poses are offered either as stereo camera pairs [3, 5], or as successive frames in a video [6, 7, 13, 16]. The supervision signal comes by using predicted depth map d_t of I_t to transform (also called warping) pixels $p \in I_t$ into the pose of the I_c, and then comparing the warped image $\widetilde{I}_c$ with I_c using photometric loss. This is a window-based loss that penalizes structural differences of the two images $\widetilde{I}_c, I_c$, meaning if the same neighborhood of pixels for the two images have similar spatial arrangements, then the loss contribution from that region is smaller. Photometric loss is defined in terms of the Structural Similarity (SSIM) and L1 loss of $\widetilde{I}_c, I_c$:

$$\mathcal{L}_{photo} = \alpha \cdot \frac{1 - \text{SSIM}(\widetilde{I}_c, I_c)}{2} + (1 - \alpha) \cdot |\widetilde{I}_c - I_c|_1,$$

where α is a relative weighting term.

2.1.1 View-synthesis pipeline

In the geometry-based view synthesis pipeline, pixels from I_t must be projected from pixel-space to 3D space, an operation called *unprojection* since projection is used to denote the reverse operation. The unprojection operation ϕ uses the predicted depth d_t and the camera's inverse intrinsics K^{-1}, where the intrinsic K is a matrix used to project points onto the image of a pinhole camera [13]. K is determined by the focal lengths f_x, f_y and pixel center c_x, c_y of the camera geometry:

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}.$$

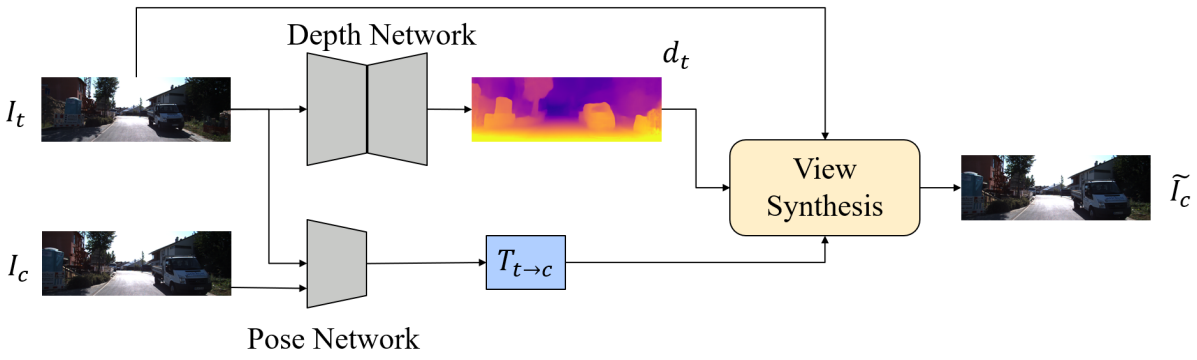


Figure 2.1: **Model architecture for view synthesis.** Given two camera images, I_t, I_c, $\widetilde{I}_c$ is synthesized from I_t, predicted depth d_t of I_t, and transformation between the views $T_{t\to c}$. Photometric loss L_{photo} compares I_c and $\widetilde{I}_c$, jointly supervising d_t and $T_{t\to c}$ (if learned).

Once in 3D space in I_t's camera reference frame, it is transformed to I_c's camera reference frame by either additionally predicting the transformation pose $T_{t\to c}$ between the frames in 3D space via a pose model that maps $(I_t, I_c) \mapsto T_{t\to c}$ or calculating it based on odometry hardware. In the case where I_t, I_c are from two separate and fixed cameras (e.g. stereo, full-surround rig [9]), camera extrinsics (the transformation from world to camera reference frame) are used instead of predicting the transformation between them. Finally, the 3D point in I_c's reference frame is projected to I_c's camera via its intrinsics K (often cameras are assumed to have identical pinhole geometries). The last step to transform pixels to their final location in the synthesized image is called projection and denoted π. The pipeline is summarized below:

1. Predict depth map d_t from image I_t using a depth network.
2. Unproject points as homogenous coordinates $p = (u, v, 1)$ in I_t to I_c's 3D reference frame using d_t and camera inverse intrinsics K^{-1}:

$$\phi(p, d_t) = d_t K^{-1} I_t(p)$$

3. Transform 3D points $\phi(p, d_t)$ to I_c's camera reference frame.

$$P = T_{t\to c}\phi(p, d_t)$$

4. Project 3D points in I_c's reference frame to I_c's camera with camera intrinsics K:

$$\widetilde{I}_c(p) = \pi(P) = \frac{1}{P_z} KP = \frac{1}{P_z} K T_{t\to c} d_t K^{-1} I_t(p)$$

The prediction image $\widetilde{I}_c$ is compared with I_c using SSIM-based photometric loss.

It is important to note that due to the geometric grounding of the view synthesis pipeline, we can generalize this process to datasets with stereo video, and even videos from multi-camera rigs. Once unprojected via ϕ, we can chain transformations between cameras *and* between time frames simultaneously (e.g. via an additional matrix multiplication step). A viewpoint is generalized to any camera image at any time, and additional supervision comes from multiple photometric losses from reconstructions between these viewpoints.

Even more surprising is that some works have shown that we can predict the camera intrinsics K if unknown [1]. Furthermore, even with cameras without pinhole geometries (e.g. fish-eye camera) or with unknown geometries, a method called NRS (Neural Ray Surfaces) [13] is used to learn the unprojection and projection operations ϕ, π themselves.

During evaluation, the depth model is considered separately to make depth predictions for single-camera images (monodepth).

2.1.2 Problems with monodepth

Photometric assumptions. There are a variety of assumptions made about the scene that when broken, can affect the quality of depth estimates. One assumption is that the scene is rigid (no dynamic objects) so that the car's ego-motion is the only moving component of the scene. Some works have been able to address this limitation by predicting dynamic motion using optical flow [1] and by masking dynamic objects completely [7]. Either way, the view synthesis pipeline becomes increasingly complex. Another assumption is about how the scene objects reflect light. non-Lambertian surfaces like mirrors or glass affect the pixel brightness differently, and depth predictions for those objects are often incorrect, as shown in Figure 2.2.

Photometric loss. The tradeoff between ground-truth LIDAR points for self-supervised photometric loss is that the latter is a weaker training signal. Since photometric loss is based on structural similarity of points around the same neighborhoods of the two photos $\widetilde{I}_c, I_c$, view synthesis operations have more leniency with a less strict loss to warp all windows of the photo correctly. Therefore, self-supervised depth estimates suffer reduced accuracy and are amenable to possible post-processing.

Scale-ambiguity for monodepth. In the monodepth view synthesis pipeline as shown in Figure 2.1, the units of scale for depth estimates are inherent to the model since nowhere are metric units reinforced. Even the pose network estimate $T_{t\to c}$ is not supervised with metric units (unless ground-truth pose from odometry is used). In other words, there is some freedom with what scale the network learns to predict depth; such methods that don't constrain the depth scale are called scale-ambiguous. When depth estimates are in a metric scale, (when metric units enter the pipeline through known, fixed camera extrinsic(s) e.g. in stereo, FSM), such methods are called scale-aware.

There are ways to alleviate scale-ambiguity. Median-scaling [14] is a method to at least match the median depth of the estimate d_t with the median depth of ground truth d^*: simply multiply ground truth estimates by a scale factor:

$$d_t^{scaled} = d_t \cdot \frac{\text{median}(d^*)}{\text{median}(d_t)}.$$

Another such method is mean-scaling, where the scale factor is $\text{mean}(d^*)/\text{mean}(d_t)$.

Still, ground truth depths are required to perform median-scaling, and even with median-scaled estimates, the model may still output different depth distributions than the ground truth distribution.

3D reconstructions. With accurate depth estimates, reconstructing the scene in 3D can still reveal "flying points", where the estimated depth at that pixel does not unproject it to the correct object as shown in Figure 2.3. In essence, this reveals the problem of depth estimates along object boundaries. Without additional processing or semantic segmentation, such boundaries present problems without explicit handling. Some works add a smoothness loss to penalize depth gradients where image sharpness decreases with an "edge-aware" term [5, 6, 13, 15, 16]. Still, recognizing and explicitly handling such points is of crucial importance when the depth network is provided as a black box for downstream applications.

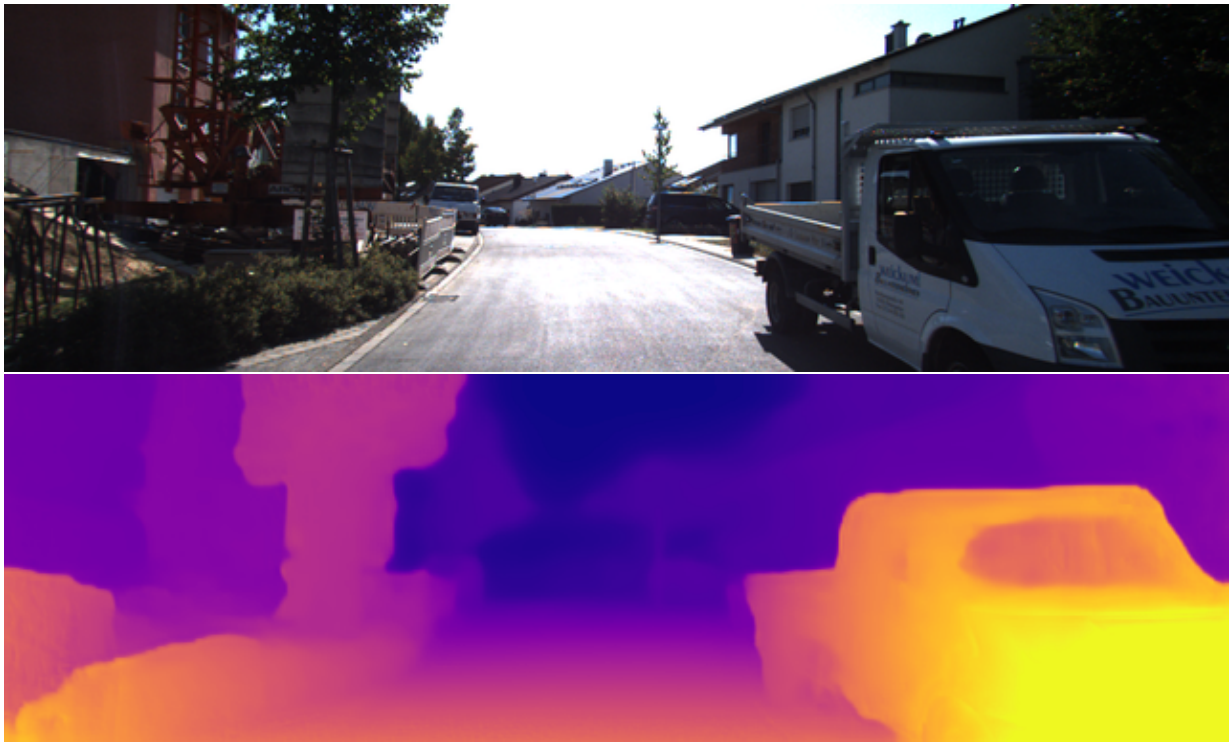


Figure 2.2: **Depth predictions on non-Lambertian surfaces.** The depth map estimate (bottom) for the camera image (top) shows that depth estimates are inaccurate for the front window (a non-Lambertian surface) of the vehicle, which illustrates a common failure case of monodepth: when photometric assumptions on which the view synthesis pipeline depend are broken.

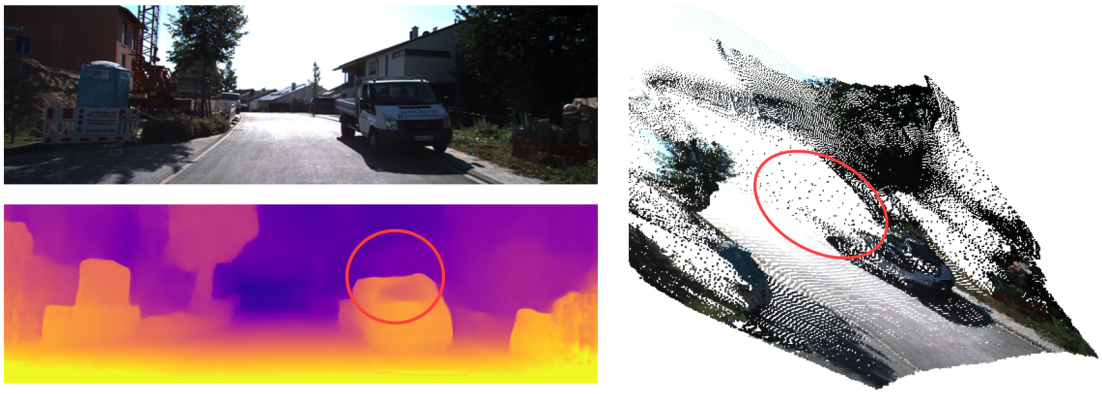


Figure 2.3: **Flying points in 3D reconstruction.** Flying points in the reconstruction (right) can be seen corresponding to the boundary of the vehicle (top left). depth map is shown bottom left.

2.2 Predictive Uncertainty

2.2.1 Black box and gray box uncertainty estimation

Black Box Uncertainty Estimation (BBUE) strategies only have access to the model as a black box. An example of a BBUE strategy is to perturb the input image and observe the change in output depth. Some perturbation T (rotation, flip, translation, etc.) is applied to input I_t, fed into the model M, then the inverse transformation T^{-1} is applied to the output $M(T(x))$. Many observations $\{(T_i^{-1} \circ M \circ T_i)(x)\}_{i=1}^{N}$ made by randomizing the perturbations T_i can be made this way. Then uncertainty is taken as the variance along the final outputs. Still, N observations cost N forward passes of the same model with N random transformations T_i, T_i^{-1}.

Gray Box Uncertainty Estimation (GBUE) strategies - such as injecting noise into model features and dropout sampling - have access to intermediate layers of the network (but not the parameters). Through this intermediate access, we can create random perturbations in a single forward pass to create the observations for statistical variance.

Feature noise: During a forward pass for depth estimation, Gaussian noise is added to the feature maps of the network's intermediate layers. In this case, each depth observation costs one forward pass, and uncertainty is again taken as the variance among the final depth outputs.

Dropout sampling. Like GBUE strategies, dropout sampling randomly affects a set of intermediate layers (sometimes just the final convolutional layer). The effect on these layers is to randomly zero out the value of certain neurons for each forward pass, as shown in Figure 2.4. Dropping a set of neurons in a forward pass forms one depth observation, so this process is repeated N times for N observations.

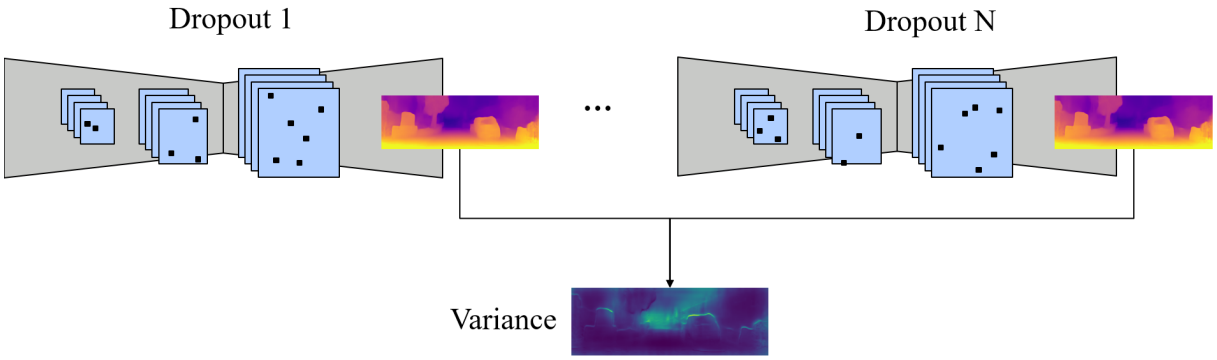


Figure 2.4: **Dropout sampling** Using random dropout to zero out certain neurons. Uncertainty is taken as variance across N independent dropout forward passes with the *same* model after training.

2.2.2 Learning uncertainty

Self-teaching. The fundamental limitation of empirical, black box, and grey-box uncertainty estimation strategies is that they require N forward passes (one for each observation). Another class of uncertainty estimation strategies, called learning or predictive uncertainty strategies, seeks to learn the uncertainty of the depth network itself with another neural network. In general, such strategies only train a network once and require only one set of models and one forward pass without any perturbations.

Poggi et. al. [11] reformulates the student-teacher framework, whereby one model (student) learns the output of another (teacher) for monodepth estimation. They call their framework self-teaching since the student network is architecturally identical to the teacher network. Assume that *given* a fixed input image I, a "teacher" depth network T has an output depth map d_T with an underlying pixel-wise distribution (assumed to be Laplacian). The "student" uncertainty network S estimates the mean and standard deviation $\mu(d_S), \sigma(d_S)$ of this distribution given I (Figure 2.5). It is important to note that S's parameter estimates $\mu(d_S), \sigma(d_S)$ are parameterized by the input. In other words, S is learning to predict T's output distribution for any given input image.

Student network S is trained to output the most probable parameters $\mu(d_S), \sigma(d_S)$ for d_t's distribution given the data (input camera image I). The formulation is the same as Maximum Likelihood Estimation (MLE), which finds the most probable parameters of some distribution

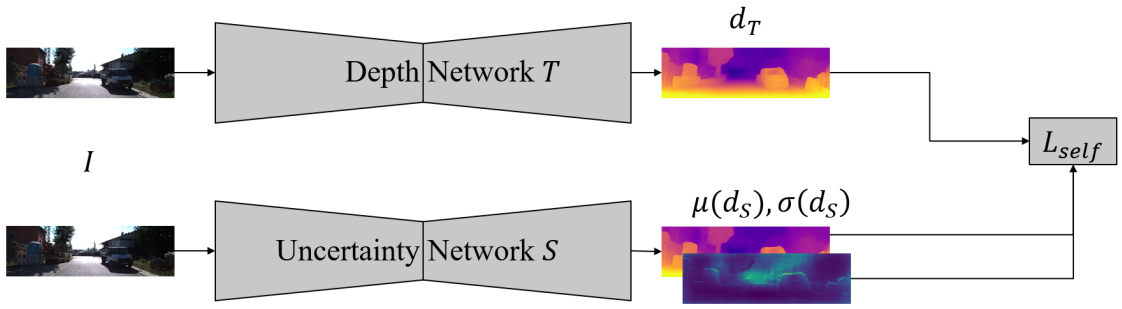


Figure 2.5: **Self-teaching.** An uncertainty network S learns to predict parameters for the output distribution of a depth network T, for any input image I. The loss L_{self} that supervises training for network S takes $d_T, \mu(d_S)$, and $\sigma(d_S)$.

family, given some fixed data/observations. Assuming a Laplacian distribution[1] on d_t [10], S is trained via minimizing the Negative Log-Likelihood, equivalently minimizing the loss:

$$\mathcal{L}_{Self} = \frac{|\mu(d_S) - d_T|}{\sigma(d_S)} + \log \sigma(d_S). \tag{2.1}$$

2.3 Empirical uncertainty

Empirical methods use many models as black box predictors and use the mean and variance of the predictions to form parameter estimates of the depth distribution. We will describe two methods that we use: bootstrap ensembles and snapshot ensembles [11].

2.3.1 Bootstrap ensembles

One straightforward way to use black box models, is train N independent depth networks. The uncertainty is defined as the variance of the independent depth estimates on a single image 2.6. However, this method requires training and storing N independent models and performing N forward passes, which is time and resource intensive.

[1]Assuming a Gaussian distribution for d_t, L2 loss instead of L1 is used in the negative log-likelihood minimization formula (Equation 2.1).

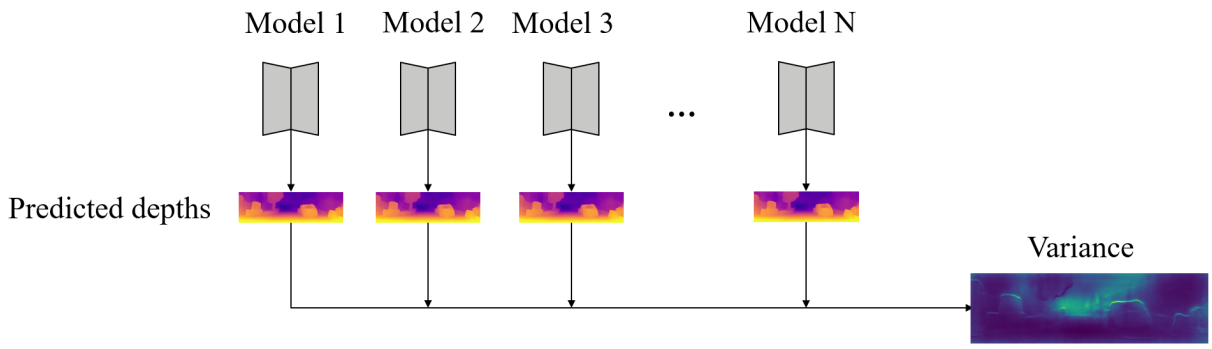


Figure 2.6: **Bootstrap ensembles.** Using variance across bootstrap ensembles as uncertainty.

2.3.2 Snapshot ensembles

Training N independent models is resource and time intensive. Instead, snapshot ensembles [11] trains a single depth network under a cyclic learning rate throughout training[2]. The model weights are saved as a single snapshot whenever the learning rate hits its lower bound. The uncertainty is defined as the variance across depth estimates of these snapshots, as shown in Figure 2.7. It is important to note that statistical independence between snapshots is an approximation since only a single training cycle is run instead of N. Still, N forward passes are needed.

[2]As a technical detail, the learning rate scheduler updates every batch instead of every epoch.

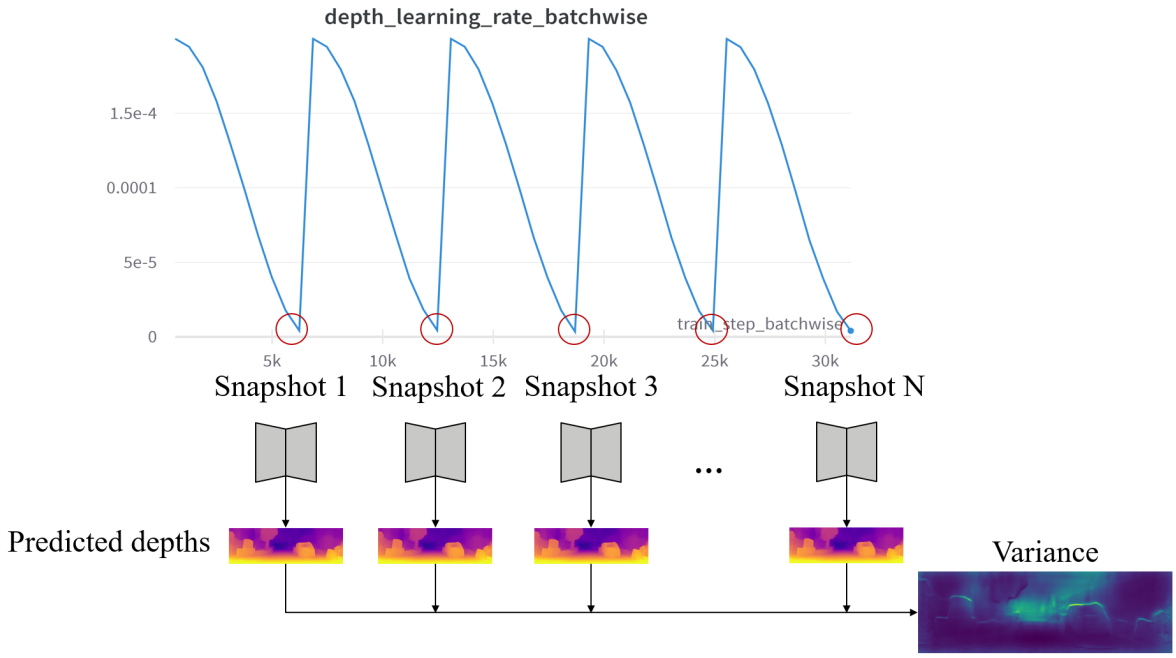


Figure 2.7: **Snapshot ensembles.** Using variance across snapshot ensembles as uncertainty. Snapshots of the model are taken at low points of a cyclic learning rate throughout a single training session.

Chapter 3

Predicting Uncertainty with Depth Scales

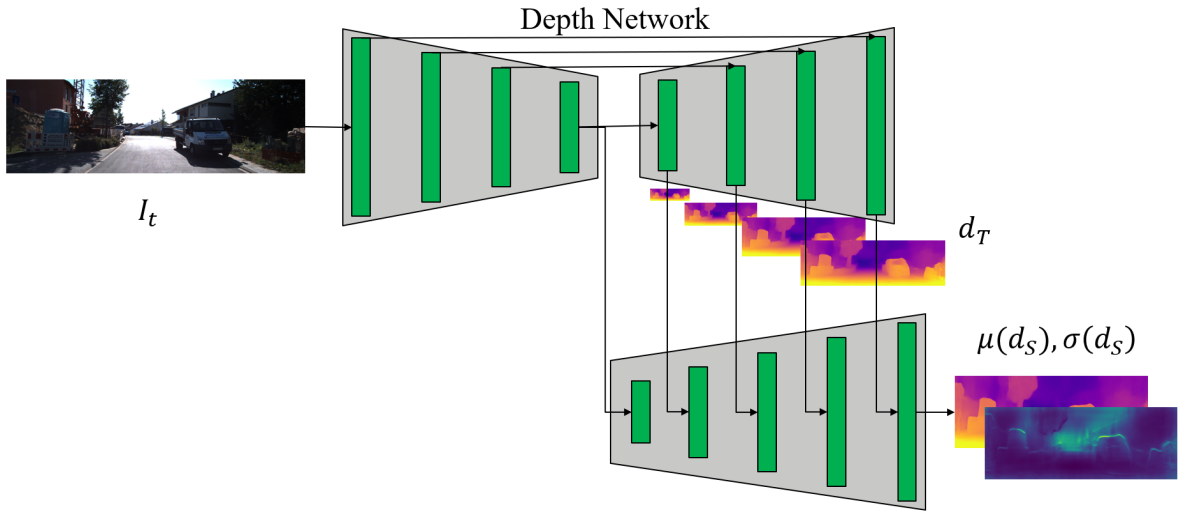


Figure 3.1: **Basic ScaleDecoder architecture.** The ScaleDecoder (bottom) takes the four depth scales of the depth decoder (top) and the U-Net bottlenet to predict parameters $\mu(d_S), \sigma(d_S)$ for d_T's underlying distribution for the given input camera image I_t. The depth scales are concatenated with the decoder's previous layer.

3.1 Uncertainty from depth scales

So far we have talked about depth networks for monodepth in general. In this work, we focus on a modified Monodepth2 [6] U-Net architecture with skip connections and a ResNet18 backbone as shown in Figure 3.2.

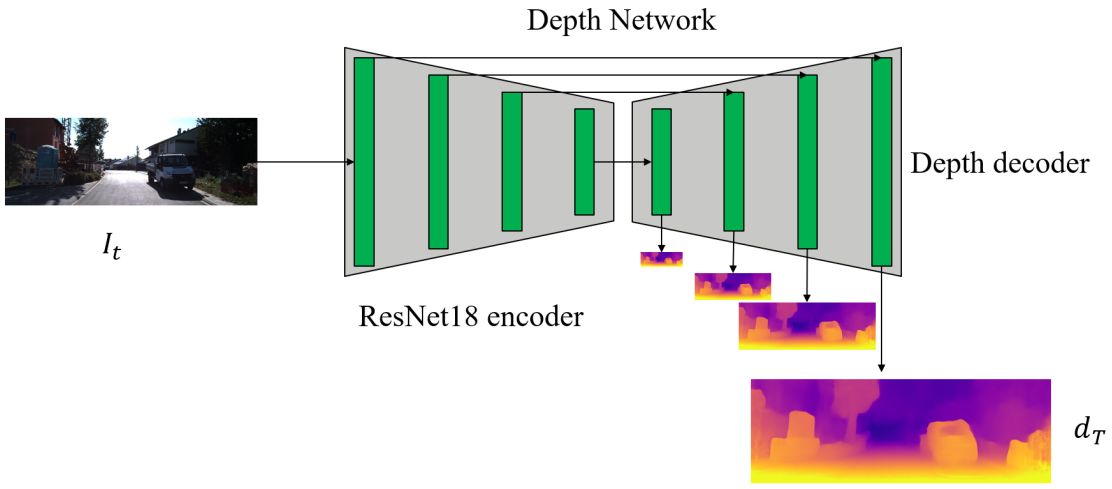


Figure 3.2: **Basic depth U-Net architecture.** The ScaleDecoder (bottom) takes the four depth scales of the depth decoder (top) and the U-Net bottlenet to predict parameters $\mu(d_S), \sigma(d_S)$ for d_T's underlying distribution for the given input camera image I_t.

Scale. Similar to how snapshot ensembles trade off the model independence assumption for more efficient training, we can treat each scale output $d_t^i,\ i \in [4]$ as a single prediction. This approach further loosens our approximation of independence. Here, uncertainty is defined as the variance across the scales[1] $\{d_t^i\}_{i=1}^4$. This uncertainty strategy needs only one training cycle and one forward pass to get all scales.

One issue with taking the variance across depth scales as the uncertainty (Section 3.1), is that each scale is twice the resolution of the previous (scales are $(H/8, W/8)$, $(H/4, W/4)$, $(H/2, W/2)$, and (H, W), where (H, W) are the dimensions of the input camera image I_t). This causes high-frequency noise in the variance as shown in Figure 3.3b.

However, the variance across depth scales (Figure 3.3b) seem to highlight the failure cases for monodepth for 3D reconstruction (Subsection 2.1.2). Not only do the boundaries of static and dynamic objects like cars have high variance, but regions such as the sky also have high variance among the scales.

(a) Camera image (b) Variance across depth scales

Figure 3.3: **Scale noise.** Uncertainty from variance across depth scales reveals high-frequency noise.

3.2 Predictive uncertainty from depth scales: novel ScaleDecoder

ScaleDecoder. As seen in the **scale** strategy, variance across depth scales seem to highlight failure cases for monodepth (e.g. inaccurate depths for object boundaries, "sky" points of infinite depth, etc.), yet contain high-frequency noise and artifacts that must be further processed. We propose a novel ScaleDecoder (Figure 3.1) to processes the U-Net bottleneck and the depth scales to produce parameter estimates for d_t's depth distribution: $\mu(d), \sigma(d)$.

ScaleDecoder Hypothesis. We hypothesize that the depth decoder scales and U-Net bottleneck feature has sufficient information to learn the depth networks output distribution. The distribution parameter $\sigma(d)$ is taken as the uncertainty and is evaluated for how well it encodes the depth errors. The details of uncertainty evaluation is described in 4.1.2.

Like the self-teaching method from Poggi et. al. [11] (Section 2.2.2), ScaleDecoder only needs to train for a single training cycle. Producing an uncertainty estimate is only a single forward pass. Furthermore, ScaleDecoder is only "one-half" of a U-Net (the decoder), requiring fewer resources to store and compute.

3.3 ScaleDecoder variants

To further understand how parts of ScaleDecoder contribute to its depth and uncertainty performance, we create variants in the training scheme, final network nonlinearity, and the number of uncertainty scales to output by ScaleDecoder.

Training scheme.

(a) *Teach* – trained and frozen depth network, only uncertainty network learns.

(b) *Detach* – depth and uncertainty networks learn together, but uncertainty gradients do not backpropogate to depth network.

(c) *Joint* - depth and uncertainty network learn together, all gradients backpropogate.

Final network nonlinearity. As in Poggi et. al. [11], the uncertainty network predicts $\log \sigma(d)$ instead of $\sigma(d)$ since it is more numerical stable and always results in positive $\sigma(d)$ by exponentiation. However, the range of the non-linearity used to predict $\log \sigma(d)$ determines the bounds for

[1]The variance is taken after "nearest"-interpolation of all scales to the highest resolution.

$\sigma(d)$. We try two different methods: the default network predicts $\log \sigma(d)$ as a fixed or learnable transformation of the final nonlinearity $\tanh$:

(a) *Default* - $\log \sigma(d) = 6 \tanh(\cdots) - 3$.

(b) *Adaptive* - $\log \sigma(d) = a \tanh(\cdots) - b$, where a, b are trainable, network parameters.

Number of output uncertainty scales.

(a) *Default* - ScaleDecoder outputs distribution parameters at a single scale $\sigma(d), \mu(d)$.

(b) *Multi-scale* - ScaleDecoder outputs distribution parameters at four scales: $\sigma_1(d), \mu_1(d)$, $\sigma_2(d)$, $\mu_2(d)$, $\cdots$, $\sigma_4(d), \mu_4(d)$.

Chapter 4

Experimental Results

4.1 Metrics

In our evaluations, we report uncertainty metrics for predictive uncertainty methods (ScaleDecoder variants) and empirical uncertainty methods (bootstrap ensembles, snapshot ensembles, and depth scales). In the depth metrics, we are evaluating the estimated mean depth $\mu(d)$ from the uncertainty strategy, not the original depth network. In the uncertainty metrics, we are evaluating the uncertainty defined for each method. Empirical strategies will use the variance across observations as the uncertainty, while predictive strategies will use the estimated distribution parameter $\sigma(d)$. The desired performance for an ideal uncertainty strategy is that it produces distribution parameters $\mu(d)$ well enough to outperform the original depth network in the monodepth task [11] and $\sigma(d)$ well enough to accurately encode depth errors (Subsection 4.1.2).

4.1.1 Depth Metrics

For evaluation, LIDAR ground truth depth map d^* is compared against the predicted depth map d_t only on valid ground-truth LIDAR pixels P. Table 4.1 shows the definitions for the metrics AbsRel (Absolute Relative), SqRel (Squared Relative), RMSE (Root Mean Squared Error), and $RMSE_{log}$ (Root Mean Squared of Log Error). During the model evaluation, these instance metrics are averaged over the entire KITTI Eigen test split [2].

So far, we have only looked at scale-variant metrics, in that the metrics depend on the scale learned by a monodepth network. We can partially address this problem by median-scaling the depth predictions by ground truth (Section 2.1.2). *After* median-scaling, the maximum scale factor between prediction d^* and ground truth d_t is defined as $\delta = \max(d_t/d^*, d^*/d_t)$. Then we can evaluate a depth map for scale accuracy as well. The notation $\delta < c$ represents the percentage of pixels p that have predicted depth $d_t(p)$ within a maximum scale factor of c with respect to ground truth $d^*(p)$ ($\frac{1}{c} < \frac{d_t}{d^*} < c$). In self-supervised depth estimation literature, c is chosen as $1.25, 1.25^2, \text{ and } 1.25^3$ for three accuracy metrics.

AbsRel↓	SqRel↓	RMSE↓	$\text{RMSE}_{log}\downarrow$
$\frac{1}{\lvert P\rvert}\sum_P \frac{\lvert d_t - d^*\rvert}{d^*}$	$\frac{1}{\lvert P\rvert}\sum_P \frac{\lvert\lvert d_t - d^*\rvert\rvert^2}{d^*}$	$\sqrt{\frac{1}{\lvert P\rvert}\sum_P \lvert\lvert d_t - d^*\rvert\rvert^2}$	$\sqrt{\frac{1}{\lvert P\rvert}\sum_P \lvert\lvert \log(d_t) - \log(d^*)\rvert\rvert^2}$

(a) Evaluation error metrics.

$\delta < 1.25 \uparrow$	$\delta < 1.25^2 \uparrow$	$\delta < 1.25^3 \uparrow$
$\max(\frac{d_t}{d^*}, \frac{d^*}{d_t}) < 1.25$	$\max(\frac{d_t}{d^*}, \frac{d^*}{d_t}) < 1.25^2$	$\max(\frac{d_t}{d^*}, \frac{d^*}{d_t}) < 1.25^3$

(b) Accuracy scale metrics.

Table 4.1: **Depth metrics' definitions.**

4.1.2 Uncertainty metrics

The principle concern in evaluating uncertainty is the lack of "ground-truth uncertainty" as a basis. Predictive uncertainty models are trained to predict the mean $\mu(d)$ and standard deviation $\sigma(d)$ of the output distribution of a depth network. Furthermore, regions of the image that have high variance of depth distribution also tend to have greater expected error with respect to ground truth depth d^*. We can thus evaluate uncertainty by measuring how well predicted uncertainty encodes the depth errors. In some sense, we want the pixels ordered by largest uncertainty to roughly match the pixels ordered by largest error. We quantify "roughly matching" these pixel orders via sparsification-based metrics AUSE and AURG [10, 11] as described below.

Sparsification. Let $P = \{p : p \in d_t\}$ be the set of pixels in a depth map estimate. Let $\epsilon : \mathcal{P}(P) \to \mathbb{R}$ be some error metric (e.g. RMSE, AbsRel, or any metric from Table 4.1) that gives a scalar value on any subset of points from P. Let ord be an ordering on the pixels (e.g. ordered by decreasing $\sigma(d)$). Let $ord[s]$ denote the set of remaining pixels P when $s \in [0, 1]$ fraction of the highest ord pixels are removed (e.g. $ord[0] = P,\ ord[1] = \varnothing$). The sparsification plot of some chosen pixel order ord (e.g. $ord = \text{decreasing } \sigma(d)$) is a measure of how ϵ evolves on $ord[s]$ as $s \to 1$ from 0.

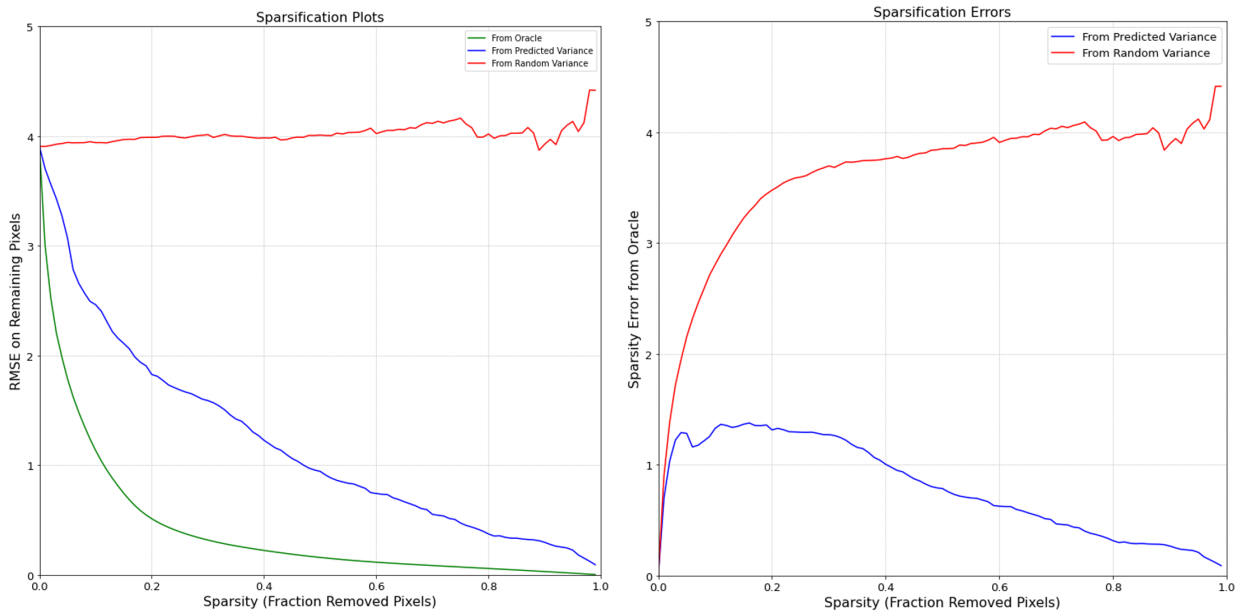
Note that in the ideal case, the pixel order given by ord is the same as the single pixels ordered by ϵ, so the sparsification plot is non-increasing. In the implementation, we iteratively (a) remove a constant fraction of the highest ord-ordered pixels (e.g. 2%), (b) plot error ϵ over the remaining set of points, and (c) repeat until no points are left.

Model sparsification. Given an error metric $\epsilon : \mathcal{P}(P) \to \mathbb{R}$ and a model's uncertainty estimate $\sigma(d)$ as the order ord. The sparsification plot for that model is a function of ϵ (y-axis) on $\sigma(d)[s]$ as $s \to 1$ from 0 (x-axis). The blue line plot in Figure 4.1a is an example of model sparsification.

Oracle sparsification. Given an error metric $\epsilon : \mathcal{P}(P) \to \mathbb{R}$, let the order ord be the ϵ order

on single pixels. The oracle sparsification is the ideal sparsification since we remove points by the highest error first. The green line plot in Figure 4.1a is an example of oracle sparsification.

Random sparsification. Given an error metric $\epsilon : \mathcal{P}(P) \to \mathbb{R}$, let the order ord be a *random* order on single pixels. The random sparsification is the weakest (expected to be a straight line as $s \to 1$ from 0) since we remove points in random order when plotting $\epsilon(ord[s])$. The red line plot in Figure 4.1a is an example of random sparsification.



(a) Example sparsification plots.

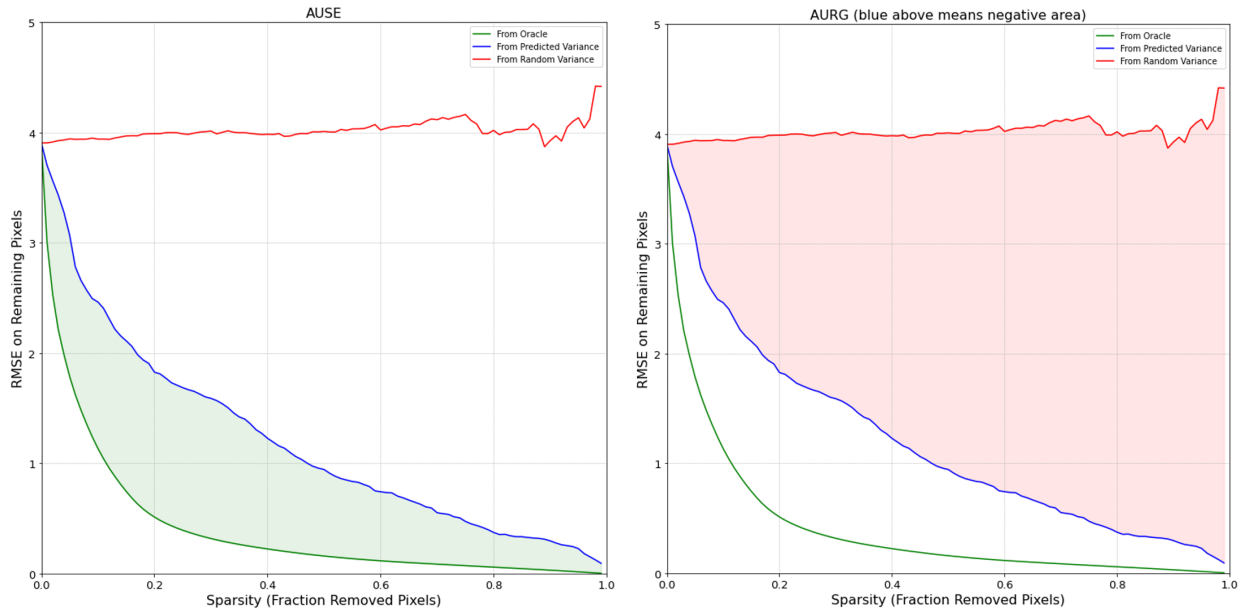
(b) Example sparsification errors.

Figure 4.1: **Example sparsification plot and errors.** In (a), the green line (bottom) is the oracle sparsification, blue (middle) is the method sparsification, and red (top) is the random sparsification. In (b), the blue (bottom) model sparsification errors and red (top) random sparsification errors result from subtracting the oracle sparsification plot from the respective sparsification plot.

Sparsification Error. To understand how well some pixel order matches the error order, the model sparsification plot needs to be compared with the oracle sparsification. We can quantify this via sparsification error: the sparsification plot in question minus the oracle sparsification, as shown in Figure 4.1b. When a sparsification error is closer to the x-axis, it's corresponding sparsification plot is closer to the oracle.

AUSE↓ To quantify "closeness to the oracle", we integrate the area between the sparsification plot in question and the oracle sparsification plot (illustrated in Figure 4.2a). Equivalently, we can integrate the area between the x-axis and the sparsification *error*, called Area Under Sparsification Error (AUSE). The closer AUSE is to 0, the closer the model sparsification is to the oracle, and the better the $\sigma(d)$ uncertainty order encodes the depth errors.

AURG↑ We can also use random sparsification as another boundary for evaluating model sparsification. A random pixel order is the most uncorrelated with the depth order, and we would like to see how much better than random order is the order from the predicted uncertainty. The Area Under Random Gain (AURG) is the area between the random and model sparsification plots



(a) Example AUSE illustrated.

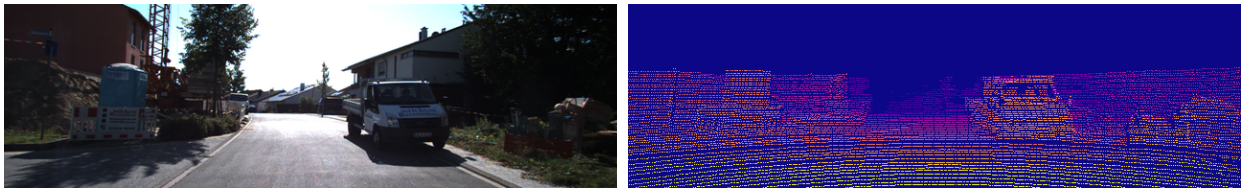
(b) Example AURG illustrated.

Figure 4.2: **Example AUSE, AURG illustrated.**

(illustrated in Figure 4.2b), or equivalently, the area between the random and model sparsification *errors*. Pixel orderings that more closely match the error order have larger AURG since the model sparsification plots are closer to the oracle and thus farther from random sparsification.

The key takeaway is that given an uncertainty estimate $\sigma(d)$ and depth errors, the AUSE↓ and AURG↑ metrics tell us how well the uncertainty encodes errors.

4.2 Data



(a) Input image.

(b) Ground truth LIDAR.

Figure 4.3: Input image (a) and Ground truth LIDAR (b) used for Figure 4.4.

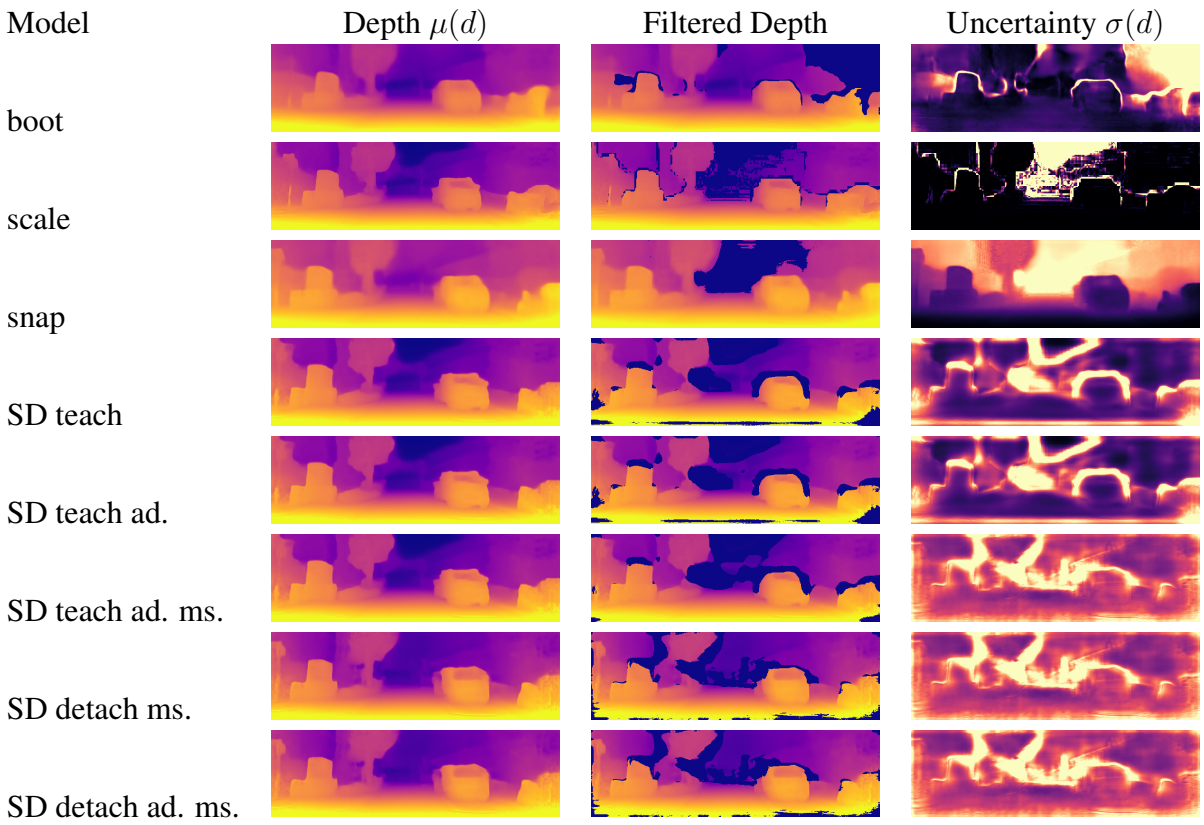


Figure 4.4: **Depth, filtered depth, and predicted uncertainty visualized.** Each row is the uncertainty inference for an uncertainty strategy. The filtered depth are the depth map d_t with 15% of the highest-uncertainty points removed. Abbreviations: "SD"=ScaleDecoder, "ad."=adaptive, "ms."=multiscale (definitions in Section 3.3).

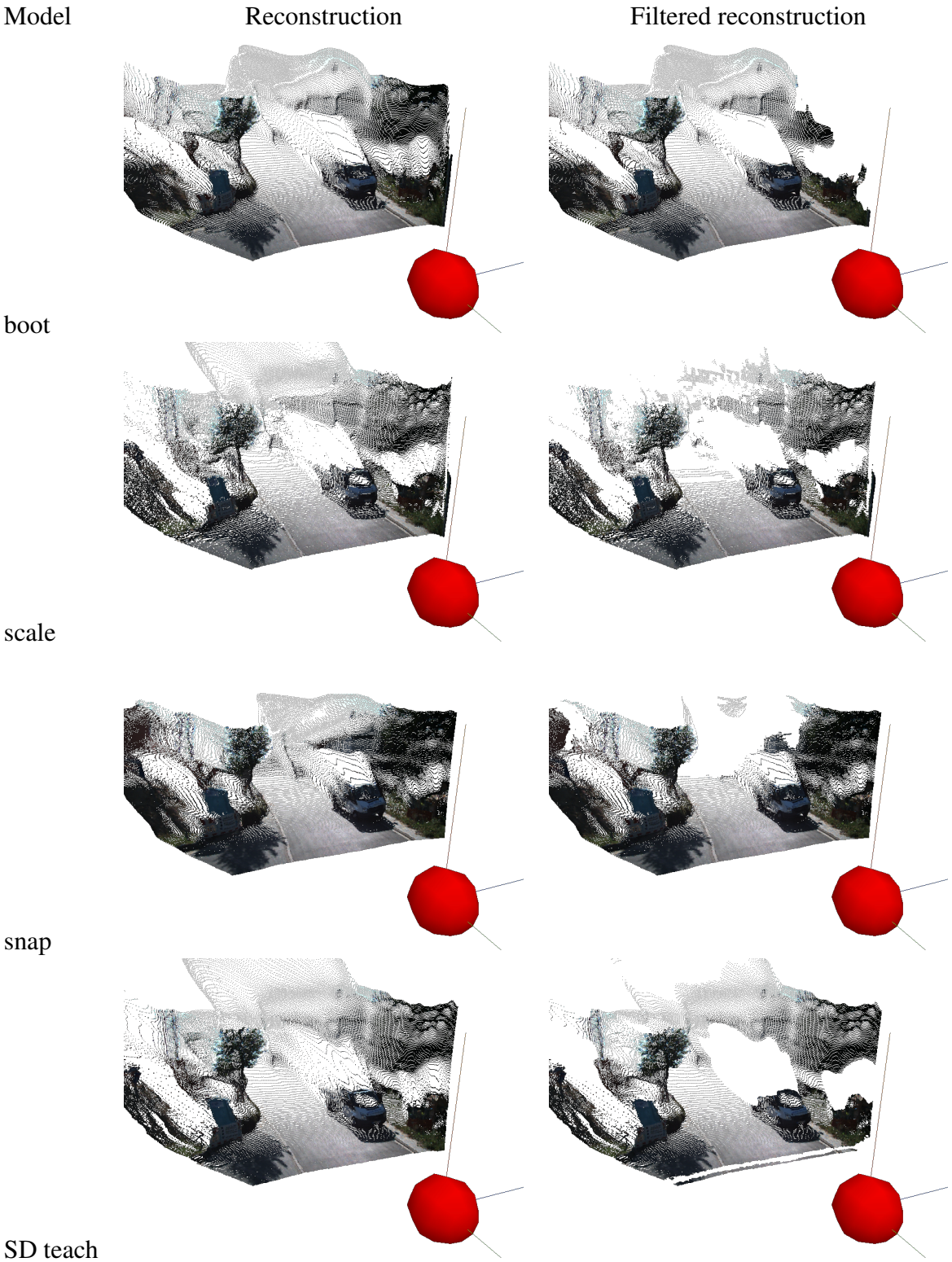


Figure 4.5: **3D reconstructions before and after uncertainty filtering.** Each row is the 3D reconstruction from depth predictions (left column) and filtered depths after removing 15% of the highest-uncertainty points. The red ball indicates the scene origin $(0, 0, 0)$.

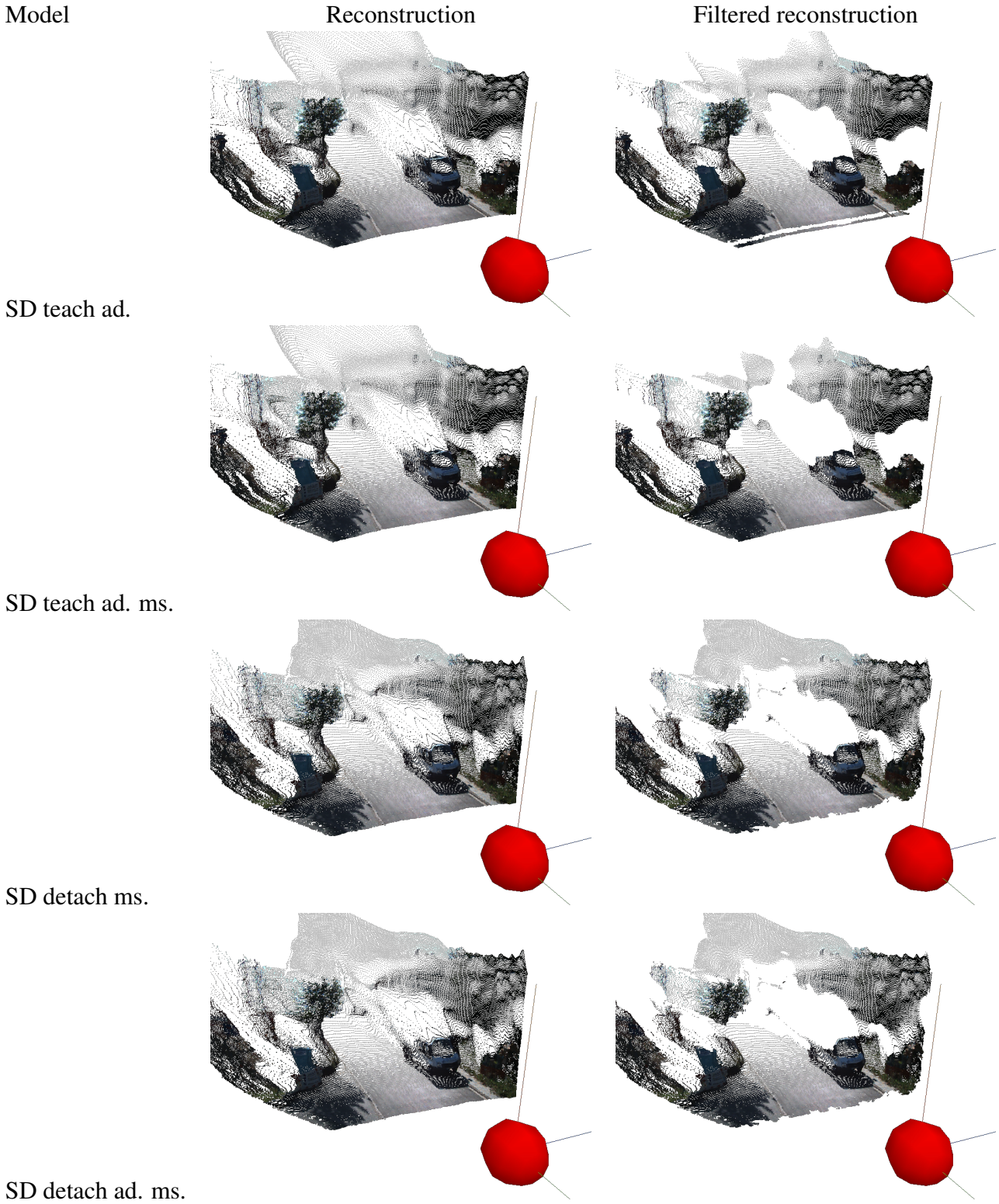


Figure 4.6: **3D reconstructions before and after uncertainty filtering.** Continuation of Figure 4.5.

	Model	AbsRel↓	SqRel↓	RMSE↓	$\mathrm{RMSE}_{log}\downarrow$	$\delta < 1.25 \uparrow$	$\delta < 1.25^2 \uparrow$	$\delta < 1.25^3 \uparrow$
depth	boot	0.973	14.849	18.939	3.633	–	–	–
	scale	0.964	14.652	18.825	3.350	–	–	–
	snap	0.974	14.963	19.063	3.688	–	–	–
	SD teach*	0.963	14.567	18.757	3.348	–	–	–
	MS SD detach	0.957	14.376	18.627	3.189	–	–	–
	MS SD detach ad.	0.956	14.366	18.638	3.163	–	–	–
	MS SD joint**	**0.922**	**13.428**	**18.034**	**2.620**	–	–	–
depth-pp	boot	0.973	14.849	18.939	3.633	–	–	–
	scale	0.964	14.656	18.828	3.351	–	–	–
	snap	0.974	14.962	19.063	3.688	–	–	–
	SD teach*	0.964	14.571	18.760	3.348	–	–	–
	SD detach MS	0.957	14.388	18.639	3.192	–	–	–
	MS SD detach ad.	0.956	14.374	18.649	3.162	–	–	–
	SD joint MS**	**0.923**	**13.455**	**18.064**	**2.620**	–	–	–
depth-gt	boot	0.126	**0.849**	**4.716**	**0.200**	0.855	**0.955**	**0.981**
	scale	**0.123**	1.048	5.087	0.204	**0.866**	**0.955**	0.978
	snap	0.189	1.549	7.681	0.287	0.691	0.888	0.954
	SD teach*	0.133	1.166	5.226	0.216	0.852	0.949	0.975
	SD detach MS	0.145	1.243	5.444	0.223	0.828	0.943	0.975
	MS SD detach ad.	0.144	1.162	5.287	0.224	0.826	0.942	0.974
	SD joint MS**	0.168	1.304	5.577	0.244	0.773	0.929	0.971
depth-pp-gt	boot	0.126	0.844	**4.704**	0.200	0.855	0.955	**0.981**
	scale	**0.119**	**0.939**	4.883	**0.199**	**0.870**	**0.957**	0.979
	snap	0.188	1.544	7.680	0.287	0.692	0.889	0.954
	SD teach*	0.129	1.052	5.035	0.211	0.855	0.952	0.977
	SD detach MS	0.144	1.126	5.283	0.220	0.825	0.947	0.977
	MS SD detach ad.	0.141	1.026	5.166	0.219	0.826	0.946	0.977
	SD joint MS**	0.159	1.146	5.348	0.235	0.784	0.939	0.975

Table 4.2: **Metrics for uncertainty-based depth predictions on KITTI Eigen test split.** The sections from top to bottom are raw depth, post-processed depth, ground-truth median-scaled depth, and post-processed ground-truth median-scaled depth. Best metrics are bolded and best metrics among the novel ScaleDecoder (SD) family are underlined. *All teaching ScaleDecoder variants have the same depth metrics. **The joint strategy reported is tested with a smaller posenet encoder.

	Model	AUSE↓	AURG↑
depth	boot	14.856	-4.971
	scale	2.851	6.923
	snap	**0.759**	**9.257**
	SD teach	7.312	2.483
	SD teach adaptive	6.923	2.873
	SD teach adaptive MS	6.157	3.640
	SD detach MS	7.896	1.828
	SD detach adaptive MS	4.322	5.425
	SD joint*	<u>3.239</u>	<u>6.211</u>
depth-pp	boot	15.155	-5.262
	scale	2.339	7.430
	snap	**0.739**	**9.280**
	SD teach	7.119	2.676
	SD teach adaptive	6.735	3.060
	SD teach adaptive MS	5.960	3.837
	SD detach MS	7.529	2.204
	SD detach adaptive MS	4.108	5.648
	SD joint*	<u>3.233</u>	<u>6.243</u>
depth-gt	boot	4.379	-0.462
	scale	1.981	2.322
	snap	**1.290**	**5.035**
	SD teach	2.452	1.947
	SD teach adaptive	2.300	2.099
	SD teach adaptive MS	1.972	2.422
	SD detach MS	2.550	1.984
	SD detach adaptive MS	1.522	2.860
	SD joint*	<u>1.470</u>	<u>3.028</u>
depth-pp-gt	boot	4.365	-0.466
	scale	1.736	2.377
	snap	**1.265**	**5.052**
	SD teach	2.299	1.926
	SD teach adaptive	2.151	2.073
	SD teach adaptive MS	1.828	2.390
	SD detach MS	2.360	1.991
	SD detach adaptive MS	<u>1.416</u>	2.835
	SD joint*	1.440	<u>2.867</u>

Table 4.3: **Metrics on uncertainty predictions.** The sections from top to bottom are raw depth, post-processed depth, ground-truth median-scaled depth, and ground-truth median-scaled post-processed depth. Best metrics are bolded. Best metric for novel ScaleDecoder (SD) among variants are underlined. *The joint strategy reported is tested with a smaller posenet encoder.

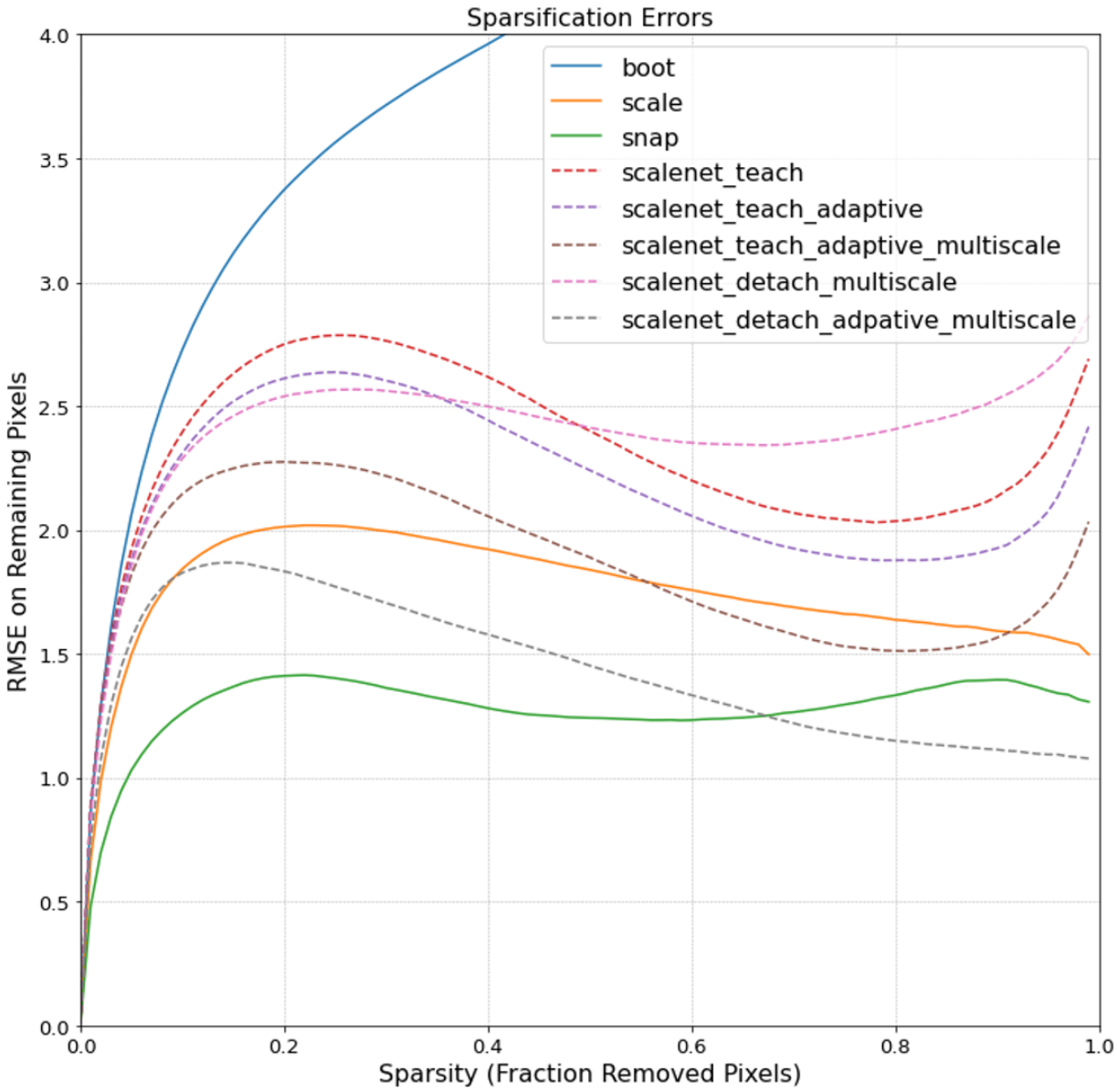


Figure 4.7: **Average test sparsification errors.** The sparsification error graphs are reported as the average over the KITTI Eigen test split [2]. We plot the RMSE metric (y-axis) on the remaining points after iteratively removing 2% of the most uncertain points (Sparsity, x-axis).

Chapter 5

Discussion

5.1 Depth

Depth metrics in Table 4.2 are evaluated with the uncertainty strategy's mean depth estimate $\mu(d)$ on the KITTI Eigen test split [2]. We focus on post-processed ground-truth median-scaled depth estimates (depth-pp-gt rows) since median-scaling allows us to compare scale accuracy δ as defined in Table 4.1b, and since depth metrics improve across the board relative to just ground-truth median-scaling (depth-gt). We report all teaching strategies together (row of SD teach*) since they have nearly identical depth metrics (all "teach" strategies estimate $\mu(d)$ from the same frozen depth network, trained from scratch).

ScaleDecoder depths generally perform comparably to empirical methods, while under-performing the boot strategy specifically. While scale is reported as the best depth strategy, it is the result of using the depth network's estimate d_t, not the uncertainty mean depth estimate $\mu(d)$. We seek to rectify this soon.

Among the ScaleDecoder variants, *teach* generally outperforms the *detach* or *joint* training schemes, regardless of number of scales or final network nonlinearity (described in Section 3.3).

We can explain the relative performance of ScaleDecoder by training schemes. The *joint* training scheme underperforms all ScaleDecoders since we are allowing gradients from the uncertainty network to "pollute" the depth network from its own task's gradients. The joint scheme not only degrades the teacher depth network whose distribution the uncertainty network is trying to learn but it changes the depth distribution itself. *Detach* training faces a similar issue in that the teacher depth network is still learning, so the training signal for the uncertainty network is weaker. This is despite the fact that the depth network's weights are detached from uncertainty network gradient updates, unlike joint training. *Teach* is the best training scheme for mean depth performance, since unlike joint or detach, the depth network is trained and frozen from scratch before the uncertainty network learns. In other words, the depth distribution has converged and is unchanging so the uncertainty network learns the distribution more accurately.

5.2 Uncertainty

Uncertainty metrics AUSE and AURG in Table 4.3 are evaluated with the uncertainty strategy's uncertainty estimate ($\sigma(d)$ for predictive models, variance for empirical models). We use $\epsilon = RMSE$ as the sparsification metric and again focus on post-processed, ground-truth median-scaled depth estimates (depth-pp-gt, last rows).

Empirical snap outperforms other methods in uncertainty estimation. As the snapshot model trains, the most severe depth errors lessen quickest as the depth network learns to minimize photometric loss across epochs. Thus, the largest source of variation across snapshots are those regions vulnerable to high error, and we can expect the variance in depth estimates to encode the largest sources of error.

Among ScaleDecoder variants, there are no clear winners when grouping by training scheme, final network nonlinearity, or the number of uncertainty scales (3.3). A comparison of all group combinations, especially after grouping by single or multi-scale predictions could reveal more patterns. Regardless, ScaleDecoder performs within the bounds of other methods (boot, snap, scale).

In Figure 4.7 we plot the sparsification errors averaged over the KITTI Eigen test split. Dashed lines are ScaleDecoder variants, and solid lines are the rest. As in 4.3, there is a similar pattern where empirical strategies form bounds for ScaleDecoder performance, and there is no clear winner among ScaleDecoders grouped by any single category (i.e. by training scheme, etc.).

While we expected self-teaching strategies to out-perform empirical ones [11], ScaleDecoder is very restrictive in that it only comprises a decoder (almost half of the uncertainty network presented in [11]) constrained to the features of another network. It performs as well as other empirical strategies, yet only requires one additional training step, one forward pass, and no feature-perturbations as described in GBUE strategies (2.2.1). ScaleDecoder thus presents as a candidate substitute for resource-heavy empirical methods that require multiple independent model training cycles and forward passes.

5.3 Qualitative visualizations

Figure 4.4 shows uncertainty estimates $\mu(d), \sigma(d)$ for each method, as well as a filtered depth row which masks out the 15% most uncertain pixels (sorted by $\sigma(d)$). In the Filtered Depth column we see that ScaleDecoder variants emphasize object boundaries as more uncertain than other methods so that filtered depth more drastically separates objects in the scene.

We can see a similar story in the 3D reconstructions and filtered reconstructions in Figure 4.5 and 4.6, where flying points are more consistently removed in the filtered 3D reconstruction around the car's roof (bottom right of scene) and pixels surrounding tree branches (top left of the scene).

5.4 Limitations of uncertainty evaluation

It is important to note that valid LIDAR ground truth points do not cover all pixels of the image (Figure 4.3b). This sparsity is an evaluation problem since uncertainty can only be evaluated by how well it encodes the depth errors which require ground truth. In other words, uncertainty evaluation as presented cannot be evaluated on invalid LIDAR pixels (e.g. "sky" regions that are infinitely far away, or non-Lambertian surfaces [glass, mirrors, car windows, etc.]), which comprise the monodepth problems (2.1.2) that we are trying to solve with uncertainty-filtering.

Another issue is the event that predictive uncertainty estimates report a region of pixels as low uncertainty when the depth at that region is incorrect. This type of uncertainty error is very dangerous since we may keep and promote erroneous points, leading to a disaster on the road with a self-driving vehicle. This possibility is more likely with predictive uncertainty since empirical models predict uncertainty with raw observation, but as Ilg et. al. concludes: "no one knows how exactly [a trained network] solves the task" [10].

Chapter 6

Future Works

We still seek to understand how much of ScaleDecoder's uncertainty performance is due to it's scale and bottleneck inputs. In other words, we would like to evaluate a ScaleDecoder variant that takes all intermediate features, and then abalate the choices of input features.

Another unsolved issue is evaluating uncertainty on pixels without ground-truth LIDAR depth since we evaluate uncertainty for how well it encodes the depth errors. Many problematic regions for the monodepth task (esp. very far away points) are invisible to the evaluation process simply because there are no valid LIDAR points for these regions. This is an important line of investigation since these are precisely the regions that may cause problems in downstream tasks.

Furthermore, we would like to compare many more uncertainty estimation strategies such as dropout, feature noise, and self-teaching [11].

Chapter 7

Conclusion

Monodepth is the problem of depth estimation from single camera images, but much interest has been sparked in trying to predict the inherent uncertainty of depth models trained in a self-supervised manner. We show various empirical, predictive, and grey-box methods for estimating the distribution of monodepth depth maps. Furthermore, we propose and evaluate ScaleDecoder to predict the depth uncertainty from the depth scales provided by Monodepth2-based architectures.

For ScaleDecoder, we hypothesize that the depth decoder scales and U-Net bottleneck provides sufficient information to learn the depth network's uncertainty. We find that ScaleDecoder performs only as well as empirical methods but with much fewer training, storage, and compute resources, presenting as a practical alternative to other empirical methods.

We also hypothesize about uncertainty filtering that if the learned uncertainty sufficiently encodes the depth errors, then we can improve overall depth accuracy by filtering the most uncertain points. Our qualitative results from ScaleDecoder suggests that "flying points" on object boundaries are removed alleviating some common problems with monodepth, leaving a cleaner 3D reconstruction from depth, but we seek to quantitatively analyze this improvement in future works.

Bibliography

[1] Yuhua Chen, Cordelia Schmid, and Cristian Sminchisescu. Self-supervised learning with geometric constraints in monocular video: Connecting flow, depth, and camera. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7063–7072, 2019. 2.1, 2.1.1, 2.1.2

[2] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *Advances in neural information processing systems*, 27, 2014. (document), 4.1.1, 4.7, 5.1

[3] Ravi Garg, Vijay Kumar Bg, Gustavo Carneiro, and Ian Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue. In *European conference on computer vision*, pages 740–756. Springer, 2016. 2.1

[4] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. 1.1

[5] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 270–279, 2017. 2.1, 2.1.2

[6] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J Brostow. Digging into self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3828–3838, 2019. 2.1, 2.1.2, 3.1

[7] Ariel Gordon, Hanhan Li, Rico Jonschkowski, and Anelia Angelova. Depth from videos in the wild: Unsupervised monocular depth learning from unknown cameras. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8977–8986, 2019. 2.1, 2.1.2

[8] Vitor Guizilini, Rares Ambrus, Sudeep Pillai, Allan Raventos, and Adrien Gaidon. 3d packing for self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2485–2494, 2020. 1.1, 2.1

[9] Vitor Guizilini, Igor Vasiljevic, Rares Ambrus, Greg Shakhnarovich, and Adrien Gaidon. Full surround monodepth from multiple cameras. *IEEE Robotics and Automation Letters*, 7(2):5397–5404, 2022. 2.1, 2.1.1

[10] Eddy Ilg, Ozgun Cicek, Silvio Galesso, Aaron Klein, Osama Makansi, Frank Hutter, and Thomas Brox. Uncertainty estimates and multi-hypotheses networks for optical flow. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 652–667,

2018. 1.2, 2.2.2, 4.1.2, 5.4

[11] Matteo Poggi, Filippo Aleotti, Fabio Tosi, and Stefano Mattoccia. On the uncertainty of self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3227–3237, 2020. 1.2, 2.2.2, 2.3, 2.3.2, 3.2, 3.3, 4.1, 4.1.2, 5.2, 6

[12] Chao Qu, Wenxin Liu, and Camillo J Taylor. Bayesian deep basis fitting for depth completion with uncertainty. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16147–16157, 2021. 1.2

[13] Igor Vasiljevic, Vitor Guizilini, Rares Ambrus, Sudeep Pillai, Wolfram Burgard, Greg Shakhnarovich, and Adrien Gaidon. Neural ray surfaces for self-supervised learning of depth and ego-motion. In *2020 International Conference on 3D Vision (3DV)*, pages 1–11. IEEE, 2020. 2.1, 2.1.1, 2.1.1, 2.1.2

[14] Lijun Wang, Yifan Wang, Linzhao Wang, Yunlong Zhan, Ying Wang, and Huchuan Lu. Can scale-consistent monocular depth be learned in a self-supervised scale-invariant manner? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12727–12736, 2021. 2.1.2

[15] Huangying Zhan, Ravi Garg, Chamara Saroj Weerasekera, Kejie Li, Harsh Agarwal, and Ian Reid. Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 340–349, 2018. 2.1, 2.1.2

[16] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1851–1858, 2017. 1.1, 2.1, 2.1.2