

Heidelberg University

Institute of Computer Engineering
Hardware and Artificial Intelligence Lab
Master's Thesis

Efficient Ensemble-based Bayesian Neural Networks for Depth Regression

Name:	Christian Simonides
Matriculation number:	3736226
First Supervisor:	Prof. Dr. Holger Fröning
Second Supervisor:	Prof. Dr. Franz Pernkopf
Date of submission:	28. May 2025

DECLARATION

I hereby certify that I have written the work myself and that I have not used any sources or aids other than those specified and that I have marked what has been taken over from other people's works, either verbatim or in terms of content, as foreign. I also certify that the electronic version of my thesis transmitted completely corresponds in content and wording to the printed version. I agree that this electronic version is being checked for plagiarism at the university using plagiarism software.

Heidelberg, 28. May 2025

A handwritten signature in black ink, consisting of a stylized 'C' followed by a series of loops and a long horizontal stroke ending in a small dot.

Christian Simonides

ABSTRACT

Depth estimation is a computer vision task that involves the estimation of distance to objects based on provided images and plays a crucial role in a wide variety of applications, such as autonomous driving. In many application areas, the decisions of Neural Networks (NNs) are safety-critical and incorrect ones can lead to serious consequences. However, traditional NNs do not provide any indication about the certainty of their predictions, which can lead to overconfidence, especially in situations that are not present in the training data. Bayesian Neural Networks (BNNs) extend the capabilities of traditional NNs by modeling uncertainty in their predictions. Since analytical Bayesian inference is often intractable for high-dimensional NNs, approximate approaches are used to realize BNNs. Monte Carlo Dropout (MCDO) and Deep Ensembles (DEs) are two BNN methods that have gained popularity due to their compatibility with established machine learning frameworks and straightforward application to NNs. Repulsive Last Layer Ensembles (RLLEs) present an alternative approach that offers the same compatibility and, additionally, focuses on resource efficiency, which is vital for complex, resource-intensive tasks such as depth estimation. To evaluate the different methods, we first perform an ablation study on a synthetic regression task and propose a new metric for epistemic uncertainty evaluation. MCDO has been found to demonstrate deficiencies in its capacity to predict epistemic uncertainties, while DEs and RLLEs both achieve good results. The application of the methods to depth estimation using established metrics yields similar results; yet they contradict the cross-validation based on artificially corrupted images. Overall, this work shows that ensemble-based BNN methods are well suited for regression tasks and even the resource-saving RLLEs perform well for the rather complex depth regression. This work is a step towards cost-efficient, generic NNs that have the ability to report uncertainty onto their predictions.

ZUSAMMENFASSUNG

Tiefenschätzung ist eine Anwendung der Computer Vision, bei der es um die Abschätzung der Entfernung zu Objekten auf der Grundlage von Bildern geht. Sie spielt eine entscheidende Rolle in einer Vielzahl von Anwendungen, wie beispielsweise beim autonomen Fahren. In vielen Anwendungsbereichen sind die Entscheidungen von Neuronalen Netzen (NN) sicherheitskritisch und fehlerhafte Entscheidungen können schwerwiegende Folgen haben. Herkömmliche neuronale Netze geben jedoch keinen Aufschluss über die Zuverlässigkeit ihrer Vorhersagen, was, insbesondere in Situationen, die in den Trainingsdaten nicht vorkommen, zur Selbstüberschätzung führen kann. Bayes'sche neuronale Netze (BNNs) erweitern traditioneller NNs um die Fähigkeit, Unsicherheiten in ihren Vorhersagen zu modellieren. Da die analytische Bayes'sche Inferenz für hochdimensionale NNs oft nicht praktikabel ist, werden approximative Ansätze zur Realisierung von BNNs verwendet. Monte Carlo Dropout (MCDO) und Deep Ensembles (DEs) sind zwei BNN-Methoden, die aufgrund ihrer Kompatibilität mit etablierten Frameworks für maschinelles Lernen und ihrer einfachen Anwendung auf NNs populär sind. Repulsive Last Layer Ensembles (RLLEs) stellen einen alternativen Ansatz dar, der auch diese Kompatibilität bietet und sich darüber hinaus auf die Ressourceneffizienz konzentriert, die für komplexe, ressourcenintensive Aufgaben wie die Tiefenschätzung bedeutend sind. Um die verschiedenen Methoden zu bewerten, führen wir zunächst eine Ablationsstudie an einer synthetischen Regressionsaufgabe durch und stellen eine neue Metrik zur Bewertung der epistemischen Unsicherheit vor. Es zeigt sich, dass MCDO Defizite bei der Vorhersage von epistemischen Unsicherheiten aufweist, während DEs und RLLEs beide gute Ergebnisse erzielen. Die Anwendung der Methoden auf die Tiefenschätzung unter Verwendung etablierter Metriken führt zu ähnlichen Ergebnissen; sie widersprechen jedoch mit einer Validierung auf der Grundlage von künstlich korumpierter Bildern. Insgesamt zeigt diese Arbeit, dass Ensemble-basierte BNN-Methoden gut für Regressionsaufgaben geeignet sind und sogar die ressourcenschonenden RLLEs bei der komplexen Tiefenregression gut abschneiden. Diese Arbeit ist ein Schritt in Richtung kosteneffizienter, generischer NNs, die in der Lage sind, Unsicherheiten in ihre Vorhersagen einzubringen.

The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge.

— Stephen Hawking

ACKNOWLEDGMENTS

First of all I would like to thank Holger Fröning for the challenging but fun journey. Many thanks also to Bernhard Klein for his kind manner and constructive support of my work. Also, many thanks to everyone else at Hawaii Labs for their support with every problem, no matter how specific. It was always enjoyable and I take a lot of good things from this time with me.

Finally, I would like to thank my family and my best friends, who have always had my back and supported me in any situation.

The author acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG.

CONTENTS

1	Introduction and Motivation	1
1.1	Contributions	2
1.2	Structure of this Work	3
2	Background	5
2.1	Bayesian Neural Networks (BNNs)	5
2.2	Monte Carlo Dropout (MCDO)	7
2.3	Deep Ensembles (DEs)	7
2.4	Repulsive Deep Ensembles (RDEs)	9
3	State of the Art and Related Work	11
3.1	Qualitative comparison of Bayesian Neural Network approaches	11
3.2	Bayesian Neural Networks for Depth Estimation	12
4	Bayesian Neural Networks applied to simple regression	15
4.1	Dataset	15
4.2	Evaluation Metrics	16
4.3	Bayesian Neural Networks on Noisy Sine	18
4.3.1	Monte Carlo Dropout (MCDO)	18
4.3.2	Deep Ensembles (DEs)	25
4.3.3	Repulsive Last Layer Ensembles (RLLEs)	30
4.4	Conclusion	36
5	Bayesian Neural Networks for Depth Estimation	37
5.1	Monocular Depth Estimation	37
5.1.1	Dataset: KITTI	38
5.1.2	Neural Architecture: Monodepth 2	38
5.1.3	Evaluation Metrics	44
5.2	Monte Carlo Dropout (MCDO)	45
5.3	Deep Ensembles (DEs)	48
5.4	Repulsive Last Layer Ensembles (RLLEs)	53
5.4.1	Activation Functions	55
5.4.2	Repulsion Strength	56
5.4.3	Ensemble Size	58
5.5	Cross-Validation using Image Corruption	59
5.6	Hardware Performance	61
5.7	Conclusion	62
6	Discussion and Outlook	65
	Bibliography	71

1

INTRODUCTION AND MOTIVATION

Conventional deep neural networks have achieved remarkable success across various domains and are now widely adopted in real-world applications [43]. Their ability to learn complex patterns from large datasets has led to significant advancements in fields such as computer vision, natural language processing, and healthcare [29]. One important task in computer vision, for example, is depth estimation, where the goal is to estimate the distance of objects in a scene from an input image [8]. Depth estimation is used in a variety of applications such as autonomous driving, robotics, and augmented reality [7]. However, the decision-making of deep neural networks is often poorly understood, as the models provide predictions without any indication of their reliability or uncertainty [23].

This can be especially problematic in safety-critical applications, such as autonomous driving, medical diagnosis, or financial forecasting, where the consequences of incorrect predictions can be severe [4]. This becomes even more problematic, if confident statements are made by the model for previously unseen scenarios and there is no way to detect or react to these uncertain predictions.

Bayesian Neural Networks (BNNs) address this limitation by providing a measure of uncertainty in their predictions and thereby allowing reasoning based on probabilities [23]. This is done by treating the model parameters as distributions rather than point estimates, enabling the model to capture uncertainty in the data and its predictions [14]. Generally, uncertainty can be split into two types: On the one hand, there is epistemic uncertainty due to insufficient data, for example because certain objects do not appear in the training data set for a classification task. On the other hand, there is aleatoric uncertainty in the training data itself, for example if a sensor captures noise during the measurements [24]. The use of probability distributions as model weights makes it possible to receive an uncertainty estimation from the BNNs, which can help to identify if the model is uncertain about a prediction. As a result, we can make more informed decisions in critical applications based on uncertainty estimates [24].

However, Bayesian inference is often intractable due to the high dimensional parameter space of neural networks [23]. Therefore, a wide variety of approximate approaches exist and can be applied in order to circumvent this issue [5, 11, 23, 24].

Monte Carlo Dropout (MCDO) estimates uncertainty by applying dropout during the inference and sampling the output distribution

using multiple forward passes [11]. It offers the advantage that it can be implemented comparatively easily using established machine learning frameworks and therefore running efficiently. Similarly, Deep Ensembles (DEs) can also be easily applied to existing models, by training multiple independent models with varying initializations. This so-called ensemble can then be used to predict uncertainty by aggregating the results of the individual models [25].

Repulsive Last Layer Ensembles (RLLEs) are a more recent approach and are based on attaching an ensemble to the output of a pre-trained model and use function space repulsion to enforce diversity among the ensemble members [5, 42]. In contrast to DEs, the ensemble only makes up a small proportion of the model, which makes the network considerably smaller. This property makes RLLEs particularly resource-efficient, which is advantageous for complex tasks.

1.1 CONTRIBUTIONS

To identify the most effective BNN approach for depth regression, this work will compare different approaches with respect to depth prediction, uncertainty prediction and runtime performance. The methods we will examine are Monte Carlo Dropout [11], DEs [25] and Repulsive Deep Ensembles (RDEs) [5] in the form of RLLEs [42].

To provide an overview of the capabilities and limitations of the approaches, we first apply them to a synthetically created Noisy Sine dataset. This enables a precise analysis of the strengths and weaknesses of each approach, because the expected prediction behavior is known. We then apply the concepts to the depth estimation task and analyze both the depth estimation performance and the uncertainty prediction. As the execution time is critical in many applications and the available hardware is often limited, the runtime performance is also measured. As the base architecture for the depth estimation we use Monodepth 2 [16] trained on the KITTI dataset [13].

In summary, the primary contributions of this work are as follows:

- Ablation study of BNN method specific hyperparameters on a synthetic dataset and introduction of a novel metric for epistemic uncertainty evaluation
- Adaptation of MCDO, DEs and RLLEs for depth estimation and the associated uncertainty prediction
- Evaluation of BNN methods for depth and uncertainty estimation and cross-validation of evaluation metrics using image corruption

1.2 STRUCTURE OF THIS WORK

In the following chapters, we will first introduce the theoretical background of BNNs and the different concepts (Chapter 2). After that we will look at overlapping and adjacent fields of research, such as other uncertainty estimation techniques (Chapter 3). In Chapter 4 we will then apply the concepts to a synthetic dataset and analyze their performance. We will then apply the concepts to the depth estimation task in Chapter 5 and compare their prediction and runtime performance. We will conclude with a summary of our findings and an outlook on future research directions in Chapter 6.

2 | BACKGROUND

This chapter provides a theoretical background to the most important concepts used in this thesis. The first section provides a general broad introduction to Bayesian Neural Networks before introducing the three main approaches used in this thesis: Monte Carlo Dropout, Deep Ensembles and Repulsive Deep Ensembles with its specialized form of Repulsive Last Layer Ensembles.

2.1 BAYESIAN NEURAL NETWORKS (BNNs)

BNNs are a category of neural networks that are based on Bayesian statistics and incorporate probabilities into their predictions. Instead of assuming the frequency of an event occurrence is fully represented in the provided data and can be described by a point estimate, BNNs work with probability distributions as model weights [24]. This enables the network to estimate the certainty of predictions by producing an output distribution rather than a single value.

In order to learn how to make predictions, BNNs update their beliefs based on their prior beliefs and observed data using the Bayes theorem

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)}, \quad (2.1)$$

where D is the observed data, H is a hypothesis, and $P(D|H)$ is the likelihood of the data given the hypothesis [23]. $P(H)$ is the prior, $P(H|D)$ is the posterior, and $P(D)$ is the model evidence or marginal likelihood, which can be rewritten as

$$P(D) = \int P(D|H)P(H)dH \quad (2.2)$$

by marginalizing over all H . The high dimensionality of the distribution space and the data makes this computation usually intractable, so we have to rely on alternative approaches [23].

Applied to BNNs the hypothesis H is equivalent to the model weights θ which means the Bayes theorem can be rewritten as

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}. \quad (2.3)$$

The target of the training process is therefore to estimate the posterior distribution $P(\theta|D)$ of the model weights θ given the observed data

D. During inference the model can then make a prediction y given an input x by marginalizing over the posterior distribution of the model weights θ :

$$P(y|x, D) = \int P(y|x, \theta)P(\theta|D)d\theta \quad (2.4)$$

As this integral is usually intractable, it is approximated by sampling from the posterior distribution of the model weights θ [23].

In order to represent the posterior distribution of the model weights, different approaches can be used. Common examples are Stochastic Variational Inference (SVI) or Markov Chain Monte Carlo (MCMC) methods. Former approximates the posterior distribution by using a simpler distribution as weights, while the latter samples from the posterior distribution directly [23]. These methods often have the disadvantage of being computationally expensive and are harder to implement compared to deterministic neural networks [25].

To capture the uncertainty of the model predictions BNNs provide an output distribution instead of a single value. Often these distributions are interpreted using the statistical metrics mean μ and variance σ^2 where a high variance represents a high predictive uncertainty [24]. By sampling outputs x_i of the distribution, μ and σ^2 can be calculated using

$$\mu = \frac{\sum_{i=0}^n x_i}{n} \quad (2.5)$$

and

$$\sigma^2 = \frac{\sum_{i=0}^n (x_i - \mu)^2}{n}. \quad (2.6)$$

The uncertainty of predictions can be divided into two categories. The first is aleatoric uncertainty, which captures the uncertainty of the training data distribution. This is mainly due to measurement errors or noise in the data and can not be reduced by collecting more data. An example would be the measurement error from a sensor adding noise to every measurement taken [24].

The second kind of uncertainty is epistemic uncertainty, which represents the uncertainty of the model due to lacking training data. This can happen, because the training data doesn't cover the whole input space, leading to regions where the model is uncertain. Therefore the epistemic uncertainty can be reduced by collecting more data [24].

To distinguish aleatoric and epistemic uncertainty a possible approach is to expand the BNN with an additional output, as the aleatoric uncertainty might be dependent on the input data. One output is therefor representing the mean of the prediction $\mu(x)$, while the second output represents the aleatoric uncertainty of the prediction $\sigma^2(x)$. The epistemic uncertainty can then be calculated from the variance of the mean output and the predicted aleatoric uncertainty [33].

The log likelihood of this approach is defined as

$$\log(P(y_i|x_i, \theta)) = -\frac{1}{2}\log(\sigma_i^2) - \frac{1}{2\sigma_i^2}(y_i - \mu_i)^2 - \frac{1}{2}\log(2\pi) \quad (2.7)$$

and should be maximized, which can be done by minimizing the negative log likelihood of the model [23]. As the last term $\frac{1}{2}\log(2\pi)$ is input independent it can be ignored. Based on the model outputs μ_i , σ_i^2 , and the target y_i the loss function is therefore defined as

$$\mathcal{L}(\theta) = \frac{1}{D} \sum_i \frac{1}{2} \sigma_i^{-2} ||y_i - \mu_i||^2 + \frac{1}{2} \log(\sigma_i^2). \quad (2.8)$$

The top term being the Mean Squared Error (MSE) divided by the variance of the predictions. The second term $\log(\sigma_i^2)$ works as a regularizer, increasing the loss for high predicted variances and therefore forcing it to smaller values [25, 33].

To make the loss function more numerical stable, the σ_i^2 is replaced by $s_i = \log(\sigma_i^2)$ resulting in the final loss function

$$\mathcal{L}(\theta) = \frac{1}{D} \sum_i \frac{1}{2} \exp(-s_i) ||y_i - \mu_i||^2 + \frac{1}{2} s_i. \quad (2.9)$$

The output therefore predicts the heteroscedastic aleatoric uncertainty $\log(\sigma_i^2)$ [24].

2.2 MONTE CARLO DROPOUT (MCDO)

One common challenge of BNNs is the additional computational cost as a trade-off for the uncertainty estimation. Monte Carlo Dropout uses dropout in the neural networks to approximate the uncertainty of predictions. Dropout is a widely used regularization technique for neural networks, that prevents overfitting by randomly setting a fraction of neuron outputs to zero during training. This forces the network to avoid the dependence on single neurons and rather learn more robust features [41].

While normal training uses dropout as a regularizer only during the training phase, MCDO applies dropout during training and during inference to approximate the uncertainty. This is done by applying a dropout layer before every weight layer and running multiple forward passes. As the dropout of the neurons is randomized, each forward pass can be interpreted as a sample of the output distribution [11].

2.3 DEEP ENSEMBLES (DES)

Lakshminarayanan et al. proposed Deep Ensembles as a type of BNN, that represent the posterior distribution of the model weights by training multiple neural networks [25]. The ensemble, which is visualized

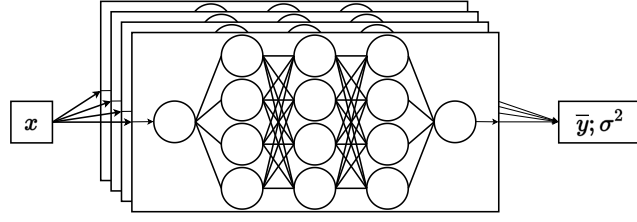


Figure 2.1: Visualization of a DE: The input x is passed through multiple independent models, which are called ensemble members. The mean and variance are calculated from the outputs of the individual models.

in Figure 2.1, consists of multiple models, called ensemble members, that are trained independently of each other.

Each model is trained on the same dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ while the initialization of the model weights is randomized, resulting in different weights for each model. Due to the different initialization, the expectation is that each member learns the target distribution in areas with a lot of data, but at the same time provides different results in areas with little data. This diversity can then be used to estimate the uncertainty of the model. The likelihood of multiple models is calculated by averaging the likelihood of the individual models using

$$p(y|x, D) = \frac{1}{M} \sum_{i=1}^M p(y|x, \theta_i) \quad (2.10)$$

where M is the number of models [25]. The learning of the model is then done by performing a maximum a posteriori (MAP) estimation and updating the weights θ_i at timestep t of a model i based on

$$\begin{aligned} \theta_i^{t+1} &\leftarrow \theta_i^t + \epsilon_t \phi(\theta_i^t) \\ \text{with } \phi(\theta_i^t) &= \nabla_{\theta_i^t} \log p(\theta_i^t | \mathcal{D}) \end{aligned} \quad (2.11)$$

with the learning rate ϵ_t [5].

As DEs are fundamentally based on traditional neural networks, they are easier to implement than other approaches and are compatible with current hardware accelerators. Meanwhile, they often achieve better performance in comparison to other Bayesian methods like MCDO [25, 44].

Nevertheless, even DEs come with the drawback of increased computational cost and an increased memory footprint, as multiple models need to be trained. This can be especially problematic for large models or datasets. As the models are trained independently of each other, the diversity between models can't be guaranteed. This can lead to feature collapse due to overparameterized models, where multiple models, even though they have individual weights, produce the same output function. This case can be problematic for uncertainty estimation as the ensemble can become overconfident [5].

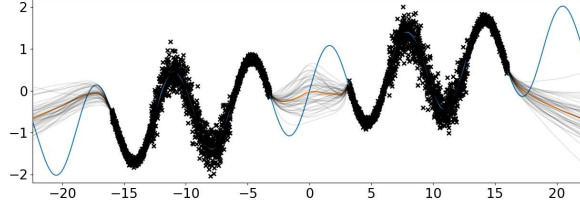


Figure 2.2: Visualization of the individual functions produced by a RDE: The ground truth is shown in blue, the training data as black marks, the individual functions of the ensemble members as gray graphs and the mean of the ensemble in orange.

2.4 REPULSIVE DEEP ENSEMBLES (RDES)

One possible solution to the problem of feature collapse in DEs is the use of Repulsive Deep Ensembles (RDEs). RDEs add interactions between ensemble members, which are also called particles, that enforces their diversity by adding a repulsion term to the loss function. This repulsion can be done either in the weight space or the function space. As the repulsion in weight space could still suffer from feature collapse [5], we will focus on the repulsion in function space.

The repulsion in function space is done by calculating the weight updates similar to Equation (2.11) with

$$f_i^{t+1} \leftarrow f_i^t + \epsilon_t \phi(f_i^t) \quad (2.12)$$

and $\phi(f_i^t) = \nabla_{f_i^t} \log p(f_i^t | \mathcal{D}) - \gamma \mathcal{R}(\{\nabla_{f_i^t} k(f_i^t, f_j^t)\}_{j=1}^n)$,

where f represents a function based on the model weight θ_i $f_i : \theta_i \rightarrow f(\cdot; \theta_i)$. The first term $\nabla_{f_i^t} \log p(f_i^t | \mathcal{D})$ is similar to the term in Equation (2.11) and is driving the model towards the target distribution. The second term $(\{\nabla_{f_i^t} k(f_i^t, f_j^t)\}_{j=1}^n)$ is the repulsion term $\mathcal{R}$, which is enforcing the function diversity of the ensemble members. This is done by subtracting the gradient of a kernel k , that is calculating the similarity between the functions f_i and the functions f_j of all other ensemble members. This result is additionally scaled by a repulsion strength factor γ . By adding repulsion, the loss and therefore gradients of the ensemble members increase if the functions of the ensemble members are too similar [5].

In regions with low density of training data, the driving term is expected to be low, which allows the repulsion term to enforce function diversity. In regions with high density of training data, the driving term is expected to be higher, and the functions can be more similar. An example visualization can be seen in fig. 2.2 where the repulsion can be seen in regions without data.

As the computation in the function space is intractable due to the high dimensionality of the input space, we instead use a set of data points to evaluate the repulsion. These data points are called repulsion samples and reduce the computational complexity of the repulsion.

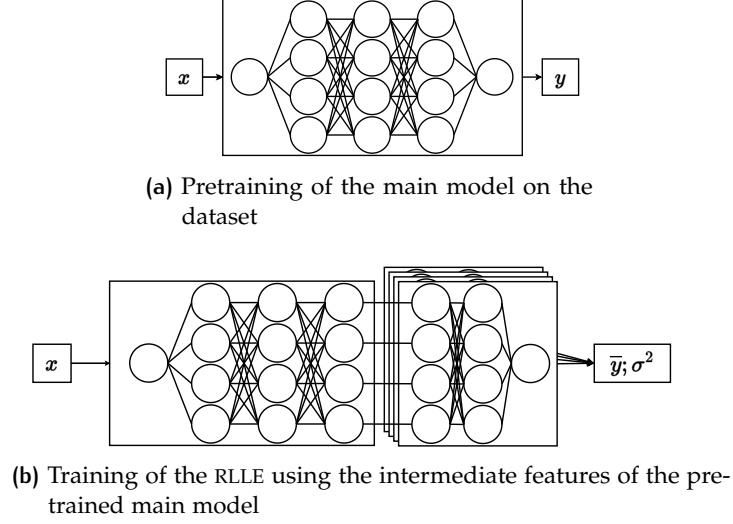


Figure 2.3: Visualization of the two training steps for RLLEs: First the main model is regularly trained on the dataset (a). The ensemble is then attached and trained using the intermediate features of the pre-trained model as inputs (b).

DEs are often used because they can be trained in parallel due to the independent model. RDEs, on the other hand, lead to better results due to the repulsion, but require an interaction between the models to calculate the diversity, which hinders parallelizability and increases the training time. Repulsive Last Layer Ensembles (RLLEs) offer a trade-off between these two approaches, by combining a neural network with a small ensemble which enforces the diversity [42].

The training of RLLEs is done in the two steps shown in Figure 2.3. First the main model is trained with the dataset using regular training techniques. In the second step, the RDE is then added using the features of the last hidden layer of the main model as inputs. Then the training of the RDE is performed like described in Section 2.4 while the main models parameters stay fixed.

3

STATE OF THE ART AND RELATED WORK

With the ever-increasing real-world use of neural networks in various fields of application, the dependence on neural networks has significantly grown in recent years. BNNs provide an approach to quantify the uncertainty of predictions, which can be important for safety-critical applications, where it is crucial to understand the confidence of a model's predictions. BNNs extend the predictions of neural networks with a confidence estimate, to better understand the prediction of a model and to make better informed decisions based on them [14].

This uncertainty estimation is particularly important in areas with high stakes like autonomous driving, where the health of users is directly at risk. In healthcare applications, for example, they can be used for the diagnosis of Alzheimer's markers on the basis of electroencephalogram signals [9], the detection of arrhythmia using electrocardiograms [2] or the analysis of electronic health records [6]. In addition, new approaches such as natural language processing with transformers are now also being researched [39].

Probably the largest application area is computer vision, which deals with the processing of images and videos. BNNs have proven to produce good results in various computer vision tasks like image classification [14] and semantic segmentation [24]. In some cases, the BNNs even outperform their non-Bayesian counterparts while adding the benefit of the uncertainty estimation.

3.1 QUALITATIVE COMPARISON OF BAYESIAN NEURAL NETWORK APPROACHES

Due to the capabilities, there is a wide range of different approaches to model uncertainty with BNNs. Each method has its own advantages, challenges and limitations, which are mainly dependent on the task used.

For example, Markov Chain Monte Carlo [17] has shown that it can be very successful in converging to a desired distribution and is often viewed as a gold standard [14]. However, due to the fundamentally different approach a prediction of the necessary iterations is difficult [1], and it also takes a comparatively long time to achieve the target distribution [12]. The non-parallelizable sequential training

is also the main disadvantage of MCMC methods, which makes them unsuitable for complex tasks and architectures.

In order to be able to apply BNNs to more complex tasks like real-time applications some research has shifted its focus to approximated approaches for BNNs. MCDO [11] offers an approach that allows the conversion of an existing architecture into a BNN by simply adding dropout layers. Even though it has shown good performance for some data sets [11], it has been shown that the predicted probability distribution does not always reflect the true posterior [46].

The general situation is similar with DEs, where evaluation results are also contradictory: DEs have shown to provide better uncertainty estimations in comparison to MCDO for some datasets [46]. However, in comparison to other approaches DEs perform comparatively poorly while they still need a long time to train [27].

RDEs and, in particular RLLEs, are relatively novel concepts. Consequently, not much research has been conducted yet, and they have been primarily compared with DEs, yielding favorable results [5, 42, 47].

3.2 BAYESIAN NEURAL NETWORKS FOR DEPTH ESTIMATION

Depth estimation is used to predict depth values for each pixel in an image [8]. While classification tasks, especially on smaller datasets like MNIST, are already well-studied using BNNs [5, 11, 42], depth estimation remains a more complex and less explored task for uncertainty estimation, which allows research to be advanced. At the same time, it is crucial for many applications like autonomous driving, robotics, or augmented reality [7].

There are numerous approaches and neural network architectures available for depth estimation, ranging from relatively simple convolutional neural networks [8] to more specialized models [10], and more recently, to the adoption of vision transformer architectures for depth estimation [3, 45]. While vision transformers currently achieve state-of-the-art performance, they come with certain drawbacks like the need for large amounts of training data and computational resources. In contrast, convolutional neural networks are more efficient and can be trained on smaller datasets, making them a more practical choice for many applications.

Some research already exists on the uncertainty prediction of depth estimation [20, 28, 31, 36]. Liu et al. proposed a video-based depth estimation method that estimates uncertainty by analyzing the temporal consistency of depth maps [28]. Poggi et al. explores uncertainty estimation techniques for depth estimation, like estimating the uncertainty by running a model once with an image and again with

a flipped version [36]. They also proposed metrics to evaluate the uncertainty of depth estimation models, which are based on the relation between the depth error and predicted uncertainty [36]. These were used by Hornauer and Belagiannis, among others, to evaluate a gradient-based uncertainty prediction [20].

While most related work propose new methods to improve uncertainty quantification, their application to depth estimation is limited. Moreover, for real world use it is unclear which of the proposed methods should be preferred considering limited computing resources while maintaining good uncertainty predictions. This work addresses this by comparing the state-of-the-art methods MCDO, DEs and as light-weight alternative RLLEs, regarding uncertainty prediction performance and computational costs.

4

BAYESIAN NEURAL NETWORKS APPLIED TO SIMPLE REGRESSION

The objective of this chapter is to gather an understanding of the impact of hyperparameters on the performance different BNNs concepts in regression tasks. An intuition for the impact of different hyperparameters is important to understand the behavior of BNNs and to be able to choose the right hyperparameters for a specific task. We therefore will start with a simple regression task, where we can easily visualize the results and directly compare the impact of the hyperparameter. For that we will evaluate the performance of MCDO, DEs and RLLEs on a synthetic dataset under varying general and concept specific hyperparameters.

4.1 DATASET

In Section 2.1 we have discussed the difference between aleatoric and epistemic uncertainty. We expect the BNN concepts to learn generalized functions from the training data, to correctly recognize the two types of uncertainties and to be able to distinguish between them. To test these capabilities, we use a dataset where we can control all the underlying parameters to evaluate the performance of the different concepts. This control allows us to define the expectations we can have about the predicted aleatoric and epistemic uncertainty of the models. A low dimensionality of the data set also allows us to observe and visualize effects in detail.

The Noisy Sine dataset is a synthetic one-dimensional dataset that contains samples of a Sine function with variable parameters. This dataset has been used by multiple authors to evaluate the performance of different methods for regression tasks [5, 42]. A major advantage is the high control over the dataset parameters, the ability to generate an arbitrary amount of data and the interpretability of the results.

In this work we are using a base function $\sin(x) + 0.05x$ with added Gaussian noise. The dataset is designed to contain regions with high epistemic and aleatoric uncertainty. This is achieved using variable density of the data points and varying the strength of added noise depending on the region. The dataset is visualized in Figure 4.1 and consists of three types of regions with different parameterizations. In the dark gray regions shown we add a Gaussian noise with a standard deviation of 0.25. We expect the model to report a high aleatoric uncertainty in these regions. In the light gray regions, the standard deviation

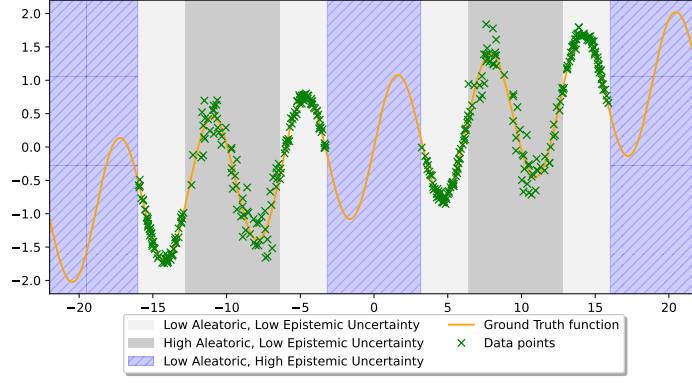


Figure 4.1: Noisy Sine dataset: The ground truth function is shown in orange, the training data in green, dark gray areas indicate regions with high noise, light gray areas indicate regions with low noise. For the blue area there are no data points given and therefore the epistemic uncertainty is high.

is 0.05 therefore we expect a lower predicted aleatoric uncertainty. As both gray regions contain data points, the epistemic uncertainty should be low. The blue regions don't contain any data points, and the model should therefore report a high epistemic uncertainty.

4.2 EVALUATION METRICS

As described in the previous section, the synthetic dataset allows us to examine precisely individual components of the model prediction. On the one hand we are interested in a generally well-fitting model that is providing good mean predictions matching the function. To quantify this prediction, we are using the two metrics Negative Log-Likelihood (NLL) and MSE. The NLL and MSE are evaluated using the ground truth data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ with N data points, and the model output $f(x_i) \Rightarrow [\mu_i, \sigma_i^2]$. Based on Equation (2.7), we calculate the negative log likelihood using

$$\text{NLL} = \frac{1}{N} \sum_i \frac{1}{2\sigma_i^2} \|y_i - \mu_i\|^2 + \frac{1}{2} \log(\sigma_i^2) \quad (4.1)$$

and the MSE is calculated by

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N \|y_i - \mu_i\|^2. \quad (4.2)$$

The lower the NLL and MSE are, the better the model fits the data.

On the other hand, we are also interested in the uncertainty estimation of the model, with a focus on the distinction between aleatoric and epistemic uncertainty. We can check the aleatoric uncertainty relatively easily thanks to the synthetically generated data: We expect the model

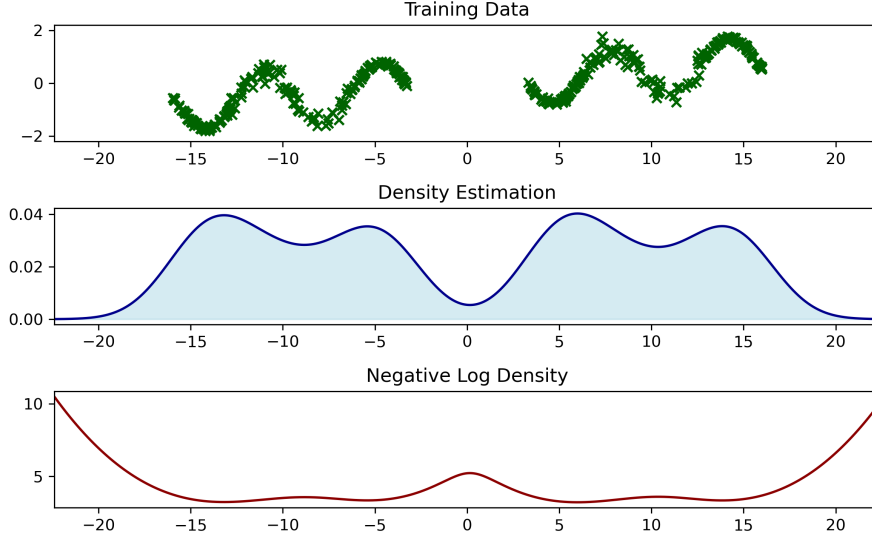


Figure 4.2: Visualization of the kernel density estimator applied to the training dataset: The training data points are shown in green on the top, the estimated density in blue in the middle and the negative log density in red on the bottom.

to predict the Gaussian noise added to the data points. For example, if the added Gaussian noise has a $\sigma^2 = 0.25$ we expect the model to also predict an aleatoric uncertainty of 0.25. The aleatoric error of the predicted aleatoric uncertainty in comparison to the ground truth variance is calculated by the MSE. A lower MSE indicates a better prediction of the aleatoric uncertainty. In regions where no training data points are given, the model cannot be expected to learn the aleatoric uncertainty, and therefore they are excluded from the calculation.

Epistemic uncertainty is more difficult to evaluate, as there is no ground truth available. We can, however, make assumptions about the expected behavior of the model. In regions with high data density, the model should predict a low epistemic uncertainty. In regions without data, the model should predict a high epistemic uncertainty, as it has no information about the function in this region. The missing ground truth also means that an absolute error of the predicted epistemic uncertainty, like the MSE, can not be calculated.

We therefore propose the Epistemic Uncertainty Coefficient (EUC) to evaluate the performance of the model in predicting epistemic uncertainty. The EUC is calculated by the Pearson correlation coefficient between the predicted epistemic uncertainty of a data point and the kernel density estimation of the input in respect to the training data.

The kernel density estimation is a non-parametric function, that estimates the probability density function by a given set of data points [35]. By smoothing with a bandwidth h over the training data points it allows us to estimate the density of the data for evaluation points. To

calculate the kernel density function at a point x given N data points $x_0, x_1, \dots, x_i$ we use the equation

$$f(x) = \frac{1}{Nh} \sum_{i=1}^N K\left(\frac{x - x_i}{h}\right) \quad (4.3)$$

with the kernel function K and the bandwidth h . As a kernel function, for example, the Gaussian kernel defined as

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \quad (4.4)$$

can be used [40].

A visualization of the kernel density estimation with $h = 2$ and a Gaussian kernel using the Noisy Sine dataset is shown in Figure 4.2. As the estimation is usually the log density, a higher value indicates a high density. By taking the negative log density we can calculate the correlation between the sparsity of the data and the predicted epistemic uncertainty. A higher value of the EUC therefore indicates a better prediction of epistemic uncertainty.

4.3 BAYESIAN NEURAL NETWORKS ON NOISY SINE

In the following section, unless stated otherwise, all models are trained with the same hyperparameters to allow a direct comparison. The training is done using the Noisy Sine dataset with 5000 training points and a batch size of 64. The model is an MLP consisting of 3 hidden layers with 50 hidden neuron and is trained with the ADAM optimizer and a static learning rate of 0.001 for 5000 epochs.

4.3.1 Monte Carlo Dropout (MCDO)

For the training with MCDO [11], we are interested in both aleatoric and epistemic uncertainty and especially their distinction. We therefore use the heteroscedastic loss function as described in Section 2.1 to train the model and allow it to predict the mean value and aleatoric uncertainty. The dropout is applied before every weighted layer, except for the first one, as dropping out a one-dimensional input value is unreasonable. For the evaluation of the activation functions the dropout rate is set to 0.1 for all layers.

4.3.1.1 Activation Functions

First, we are investigating the impact different activation functions have on the performance of MCDO. Neural networks often perform vastly differently depending on the activation function used, and a good choice can significantly improve the performance. The results

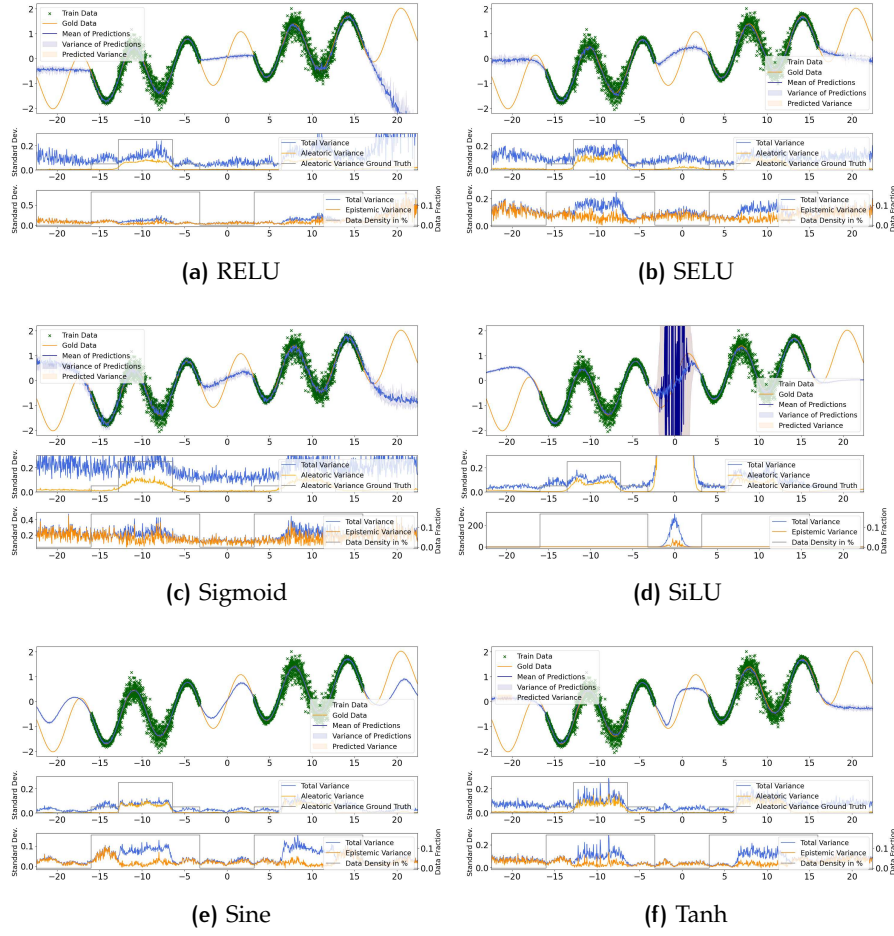


Figure 4.3: Comparison of different activation functions for MCDO: All functions are able to fit the training data, but the SiLU function shows an instability in the region around $x = 0$. The uncertainty predictions vary significantly between the different activation functions.

Act	NLL ↓	MSE ↓	Aleatoric Error ↓	EUC ↑
RELU	0.946	1.904	0.016	<u>0.587</u>
SELU	0.374	0.615	0.015	0.329
Sigmoid	0.541	1.071	<u>0.014</u>	0.292
SiLU	3.665	6.966	0.015	0.083
Sine	<u>0.120</u>	<u>0.219</u>	0.016	0.013
Tanh	0.324	0.626	<u>0.014</u>	0.419

Table 4.1: Comparison of metrics for different activation functions for MCDO: The Sine activation function performs best in terms of NLL and MSE, while the Tanh activation function achieves the best Aleatoric MSE and the RELU the best EUC score.

for six common activation functions using the above-mentioned hyperparameters are shown in Figure 4.3 and Table 4.1. The upper part of each figure shows the plot of the ground truth function in orange and the training data points in green. The blue line shows the mean prediction of the outputs, and the orange shaded area the predicted aleatoric uncertainty. The blue shaded area shows the total variance of the model prediction. In the middle part of each figure the total variance is plotted in blue and the predicted aleatoric variance in orange. The ground truth aleatoric variance is shown in black, where we expect the model to predict high aleatoric uncertainty. The bottom plot again shows the total variance in blue, and the epistemic uncertainty in orange. For a high data density, we expect the model to predict low epistemic uncertainty and for regions without data a high epistemic uncertainty.

Figure 4.3 shows that all activation functions are generally able to fit the training data. This is also emphasized by the low NLL and MSE values in Table 4.1. One exception is the SiLU function in Figure 4.3d which has a noticeable instability in the region around $x = 0$. This also results in significantly higher NLL and MSE. The best performing activation function in terms of NLL and MSE is the Sine.

The graphs in the middle reveal that all activation functions are also able to predict the aleatoric uncertainty in the regions with high noise. In the regions where the aleatoric ground truth is high, meaning where the black graph shows a high variance, all models indicate aleatoric uncertainty, shown by the orange graph. The aleatoric error in Table 4.1 of all activation functions is at a similarly low level.

The predicted epistemic uncertainty shown in the bottom graph in orange, reveals that the different activation functions perform very differently. There are three regions without data: the left, the middle and on the right. As an example, RELU correctly predicts a high epistemic uncertainty in the right region and some in the left region, but no significant epistemic uncertainty in the middle region. SELU on the other hand predicts a high epistemic uncertainty in all three regions, but also some in regions where data is given. The Sine activation function predicts the highest epistemic uncertainty at $x = \{-15; 15\}$, which is not expected and results in the lowest EUC.

Combining the results of the NLL, MSE, aleatoric error and EUC we will use Tanh as the activation function for the following experiments. It provides a balanced performance with a low NLL and MSE, the lowest aleatoric error and the second highest EUC.

An important fact that should not be overlooked is that Sine as an activation function is directly related to the test data set. The ground truth data is generated from a Sine function, which could explain the very good results for the mean prediction. At the same time, however, the periodicity of the Sine could lead to epistemic uncertainty being

λ_{reg}	NLL ↓	MSE ↓	Aleatoric Error ↓	EUC ↑
0.10	1.903	3.846	0.131	0.707
0.25	0.641	1.290	0.012	0.664
0.50	0.708	1.385	0.008	0.640
1.00	0.324	0.626	0.014	0.419
2.00	0.162	0.312	0.020	0.294
5.00	0.121	0.237	0.024	0.193

Table 4.2: Comparison using different variance regularizer values for MCDO: Using a variance regularizer of 0.5 yields the best aleatoric uncertainty prediction.

underestimated. For these reasons, we do not use it for comparative purposes.

4.3.1.2 Variance Regularizer

A major issue all previous results, independent of the used activation function, face is that the predicted variance is significantly lower than expected. As described in Section 2.1 we are using the loss function

$$\mathcal{L}(\theta) = \frac{1}{D} \sum_i \frac{1}{2} \exp(-s_i) \|y_i - \mu_i\|^2 + \frac{1}{2} s_i. \quad (4.5)$$

As the predicted aleatoric uncertainty $s_i = \log(\sigma_i^2)$ is lower than expected, the suspicion is the regularizer term $\frac{1}{2}s_i$ forces the model to predict a lower uncertainty. By adding a hyperparameter λ_{reg} we can control the impact of the regularizer term on the loss, which should allow the model to predict higher variances. The loss function is therefore changed to

$$\mathcal{L}(\theta) = \frac{1}{D} \sum_i \frac{1}{2} \exp(-s_i) \|y_i - \mu_i\|^2 + \lambda_{reg} \frac{1}{2} s_i, \quad (4.6)$$

with λ_{reg} as a variance regularizer. We can therefore vary this hyperparameter to minimize the aleatoric MSE and achieve the best prediction of aleatoric uncertainty.

In Figure 4.4 and Table 4.2 the results for six different variance regularizer values are shown. The results reveal a direct correlation of the variance regularizer on the predicted variance. The smaller the hyperparameter λ_{reg} is chosen the higher, the higher is the predicted variance. The Figure 4.4c reveals, that the predicted aleatoric uncertainty for $\lambda_{reg} = 0.5$ is a little lower than the expected variance, while in Figure 4.4b with $\lambda_{reg} = 0.25$ the predicted variance is higher than expected.

Using very low values for the variance regularizer, like $\lambda_{reg} = 0.1$ also results in some predicted variance in regions, where it should be

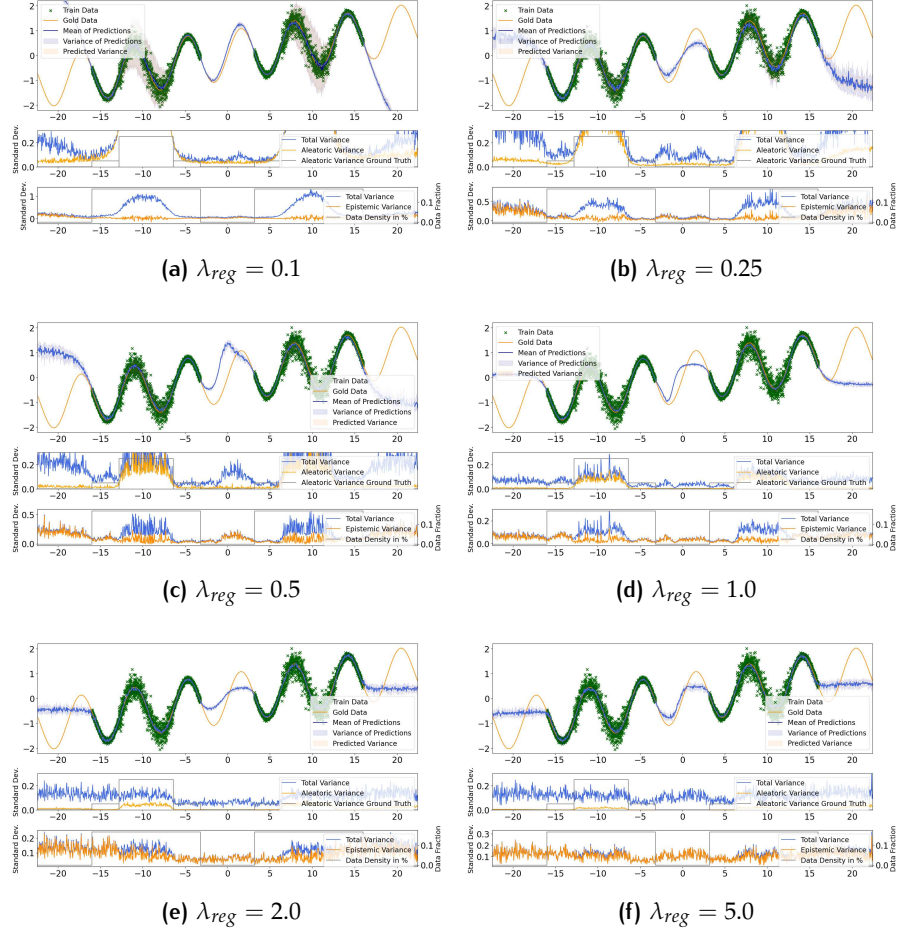


Figure 4.4: Comparison of different variance regularizers for MCDO. The orange graph in the middle plot shows that the predicted aleatoric uncertainty is directly impacted by the hyperparameter λ_{reg} . Using $\lambda_{reg} = 0.1$ (a), the model clearly overestimates the aleatoric uncertainty, while at $\lambda_{reg} = 5$ (f) almost no aleatoric uncertainty is predicted.

Dropout Rate	NLL ↓	MSE ↓	Aleatoric Error ↓	EUC ↑
0.01	0.216	0.393	0.008	-0.025
0.05	0.382	0.740	0.008	0.346
0.10	0.708	1.385	0.008	0.640
0.15	0.169	0.278	0.007	0.241
0.20	0.412	0.790	0.008	0.179
0.30	0.399	0.753	0.007	-0.126

Table 4.3: Comparison of the performance MCDO with different dropout rates: A dropout rate of 15% results in the best NLL and MSE, but the epistemic uncertainty prediction is poor. With a dropout rate of 10%, the NLL and MSE are worse, but the EUC is significantly better.

low. Comparing the results of Figure 4.4a and Figure 4.4d we can see that the predicted variance for the outer regions is higher for $\lambda_{reg} = 0.1$ than for $\lambda_{reg} = 1.0$. This predicted aleatoric variance causes the model to predict a lower epistemic uncertainty in the outer regions, which is not desired.

One important caveat is the impact of the variance regularizer on the other metrics. As we increase the value of the variance regularizer, the values of the NLL and MSE also increase, implying a worse performance. A lower regularizer value results in greater variance when sampling the model outputs. However, since the value for the NLL and MSE are calculated on the ground truth function, the wider output distributions inherently worsens the NLL and MSE values.

The EUC value is also dependent on the hyperparameter choice, as only a correctly predicted aleatoric uncertainty allows the model to distinguish between aleatoric and epistemic uncertainty. The best performing hyperparameter for the variance regularizer is $\lambda_{reg} = 0.5$ as it provides a good balance between the aleatoric error and the EUC. For this reason, we will use $\lambda_{reg} = 0.5$ for the following experiments.

4.3.1.3 Dropout Rate

The final hyperparameter we will investigate is the dropout rate of the model during training and inference. The dropout rate is a crucial hyperparameter for MCDO as it controls the amount of noise added to the model. We apply the dropout with the specified rate before every weighted layer, except for the first one.

The results for different dropout rates between 0.01 and 0.3 are shown in Figure 4.5 and Table 4.3. For all dropout rates the model is able to fit the training data well and predict the aleatoric uncertainty almost equally good, which is also evident from the low aleatoric error values in Table 4.3. The graphs reveal that the predicted aleatoric

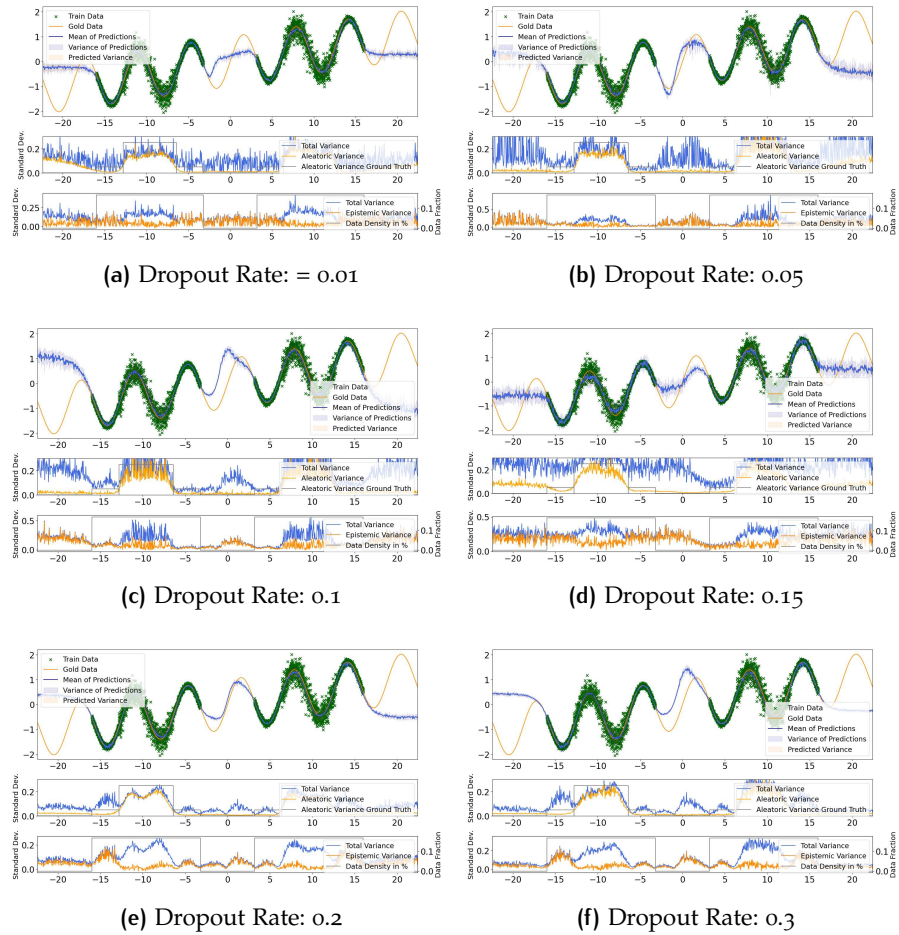


Figure 4.5: Comparison of different dropout rates for MCDO: For a dropout rate of 10% (c), the bottom graph shows the expected increase in the predicted epistemic uncertainty in the outer and middle region. If the dropout rate set to 15% (d), this increase becomes much less noticeable.

uncertainty is generally tending to be lower than expected, but the noise, especially for 0.1 seems to obscure this effect.

For the dropout rates of 0.01 and 0.15 the predicted aleatoric uncertainty in the outer regions, where no training data is given, starts to increase. However, this behavior does not seem to have a direct correlation with the dropout rate, as dropout rates between these two, as well as outside them, do not show this behavior.

The epistemic uncertainty seems to tend to get worse, the further the dropout rate is from the original 0.1 used in the previous experiments, where the other hyperparameters were optimized. For a dropout rate of 0.1 the EUC indicates the best results, for 0.05, 0.125 and 0.2 the EUC is at a worse level. For 0.01 and 0.3 the EUC is gets negative, which means that the model tends to predict a higher epistemic uncertainty in regions with training data than in regions without data.

Given the results shown, we can draw the conclusion, the variance regularizer and dropout rate should be tuned to each other to achieve the best result. This is the only way for the model to predict a good aleatoric uncertainty, as well as a good epistemic uncertainty. In general dropout rates between 0.05 and 0.2 seem to be a good choice as a starting point, as they all provide acceptable results.

4.3.2 Deep Ensembles (DEs)

Similar than for MCDO we are going to investigate the impact of different hyperparameters on the performance of DEs [25]. The aim here is again to check whether and under what conditions DEs can correctly predict uncertainties and whether the separation between aleatoric and epistemic uncertainty is correct. The initial values of the hyperparameters and the model architecture are the same as for MCDO. As DE does not use a dropout rate, this value is therefore set to 0. Instead of the dropout, for DEs the size of the ensemble is relevant and will be set to 10 unless otherwise specified.

4.3.2.1 Activation Functions

For the DEs, we also first investigate the performance of different activation function to ensure optimum conditions for the following experiments. The results can be seen in Figure 4.6 and Table 4.4. Some anomalies in the graph become quickly apparent: Similar to the SiLU function for MCDO in Figure 4.3d, the SELU function becomes unstable in the outer regions. This also leads to a very high NLL and MSE compared to the other results. The second anomaly applies to the Sigmoid activation function, which also fails in this case to predict the aleatoric and epistemic uncertainty. Even if the outer regions show an increased variance of predictions, the uncertainty predictions are not usable in data-rich regions. Due to the consistently high aleatoric

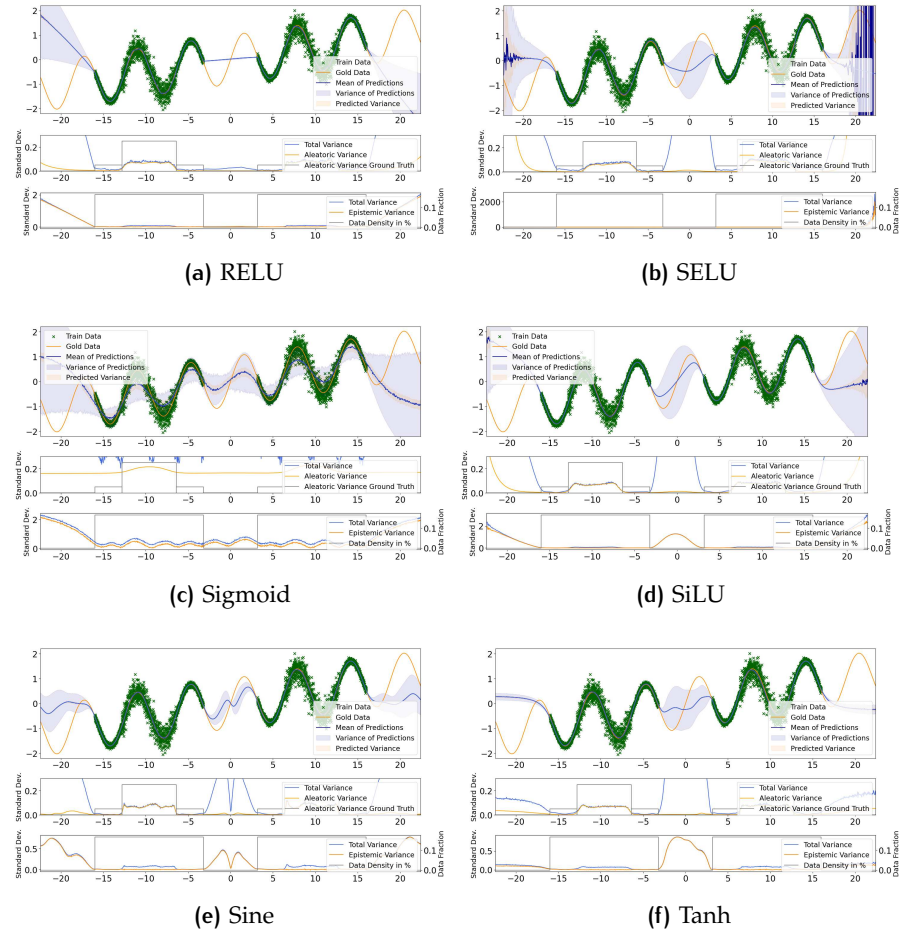


Figure 4.6: Comparison of different activation functions for DEs: All activation functions learn the ground truth function, except for the sigmoid (c) function, which struggles doing so. The SELU (b) becomes unstable in the outer regions.

Act	NLL ↓	MSE ↓	Aleatoric Error ↓	EUC ↑
RELU	0.959	1.985	0.016	<u>0.949</u>
SELU	6.292	286.123	0.016	0.471
Sigmoid	0.560	1.116	<u>0.009</u>	0.907
SiLU	0.500	1.070	0.016	0.943
Sine	<u>0.216</u>	<u>0.411</u>	0.016	0.837
Tanh	0.342	0.677	0.016	0.216

Table 4.4: Performance of different activation functions using DE: The Sine activation function best fits the training data, while RELU achieves the highest EUC.

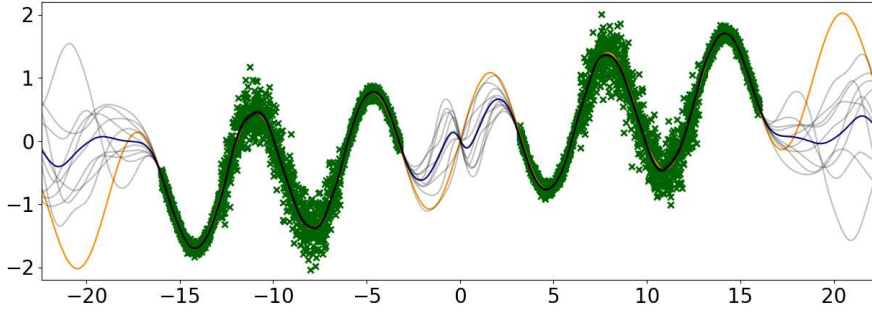


Figure 4.7: Plot of the individual particles of a DE for the Sine function: Training data in green, ground truth in orange, predicted mean in blue, predictions of the individual models in gray. At $x = 0$ all the models are predicting a similar value, which leads to a low uncertainty.

uncertainty prediction, Sigmoid produces the lowest aleatory error, even if the prediction is not as expected.

Additionally, the Sine function shows a conspicuous behavior for the epistemic uncertainty prediction at $x = 0$, where the epistemic uncertainty drops significantly. To investigate this further, we can take a look at the plot of the output functions of the individual models that are part of the ensemble in Figure 4.7. It is apparent that the individual models of the ensemble are diversifying in regions without data, which is the desired behavior. However, due to the periodic nature of the Sine function, all ensemble members tend to go through the origin. Since the models all predict similar values at this point and are in mutual agreement, this results in a very low epistemic uncertainty in this region, which is not desired. The different initialization of the particles of the ensemble members does not seem to be sufficient to prevent this behavior.

Fundamentally all activation functions, except for sigmoid, perform decently and are able to fit the training data. The uncertainty graphs in Figure 4.7 show, that the aleatoric portion is similarly well detected by all activation functions, which is also evident in the aleatoric error in Table 4.4, although the values are predicted too low. We will therefore explore the influence of the variance regulator on the results. The epistemic uncertainties are also predicted well by all activation functions, although the Tanh predicts a way lower epistemic uncertainty in the outer regions than in the middle region. Since the SiLU activation function provides good results for the NLL, MSE and a good EUC value, we will use this function in the following experiments.

4.3.2.2 Variance Regularizer

Since DEs are affected by too low predictions of aleatoric variance similar to MCDO, we are investigating the impact of the variance regularizer on the results. The results for different values of the variance

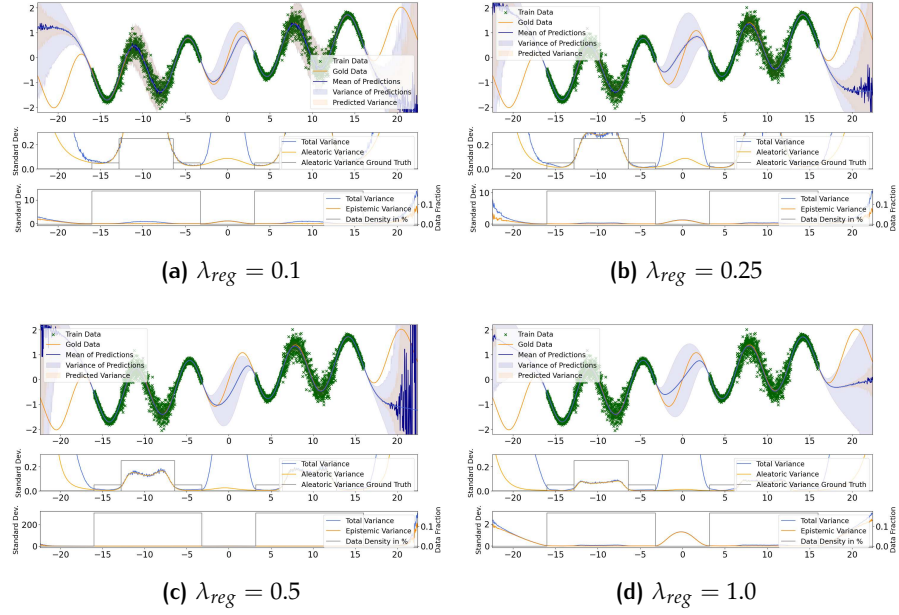


Figure 4.8: Comparison of different variance regularizers for DEs. With a λ_{reg} of 0.25, the prediction of aleatoric uncertainty is the most accurate, as it is only slightly overestimated.

λ_{reg}	NLL ↓	MSE ↓	Aleatoric Error ↓	EUC ↑
0.10	0.613	1.921	0.102	0.870
0.25	0.586	1.934	0.007	0.914
0.50	1.410	13.406	0.009	0.539
1.00	0.500	1.070	0.016	0.943

Table 4.5: Performance of the DE model with different variance regularization values: The lowest aleatoric error is achieved for $\lambda_{reg} = 0.25$.

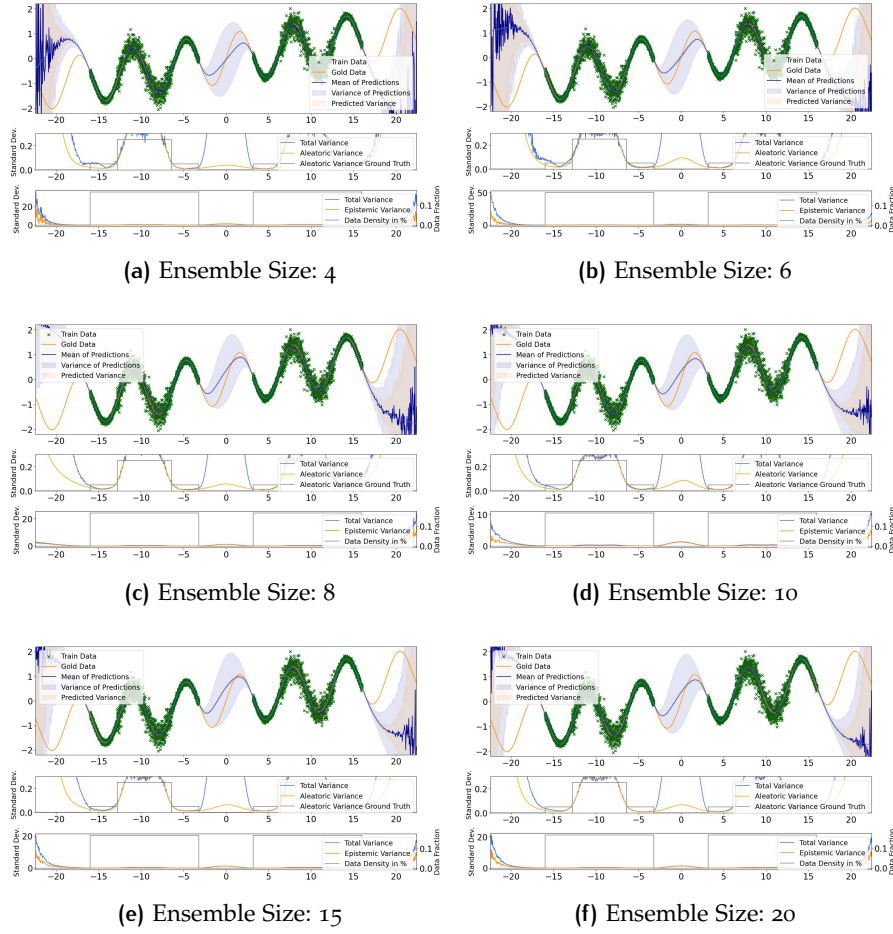


Figure 4.9: Comparison of different ensemble sizes for DE. The mean and uncertainty predictions are generally similar for all ensemble sizes. However, ensembles become more stable the larger they are chosen.

regularizer are shown in Figure 4.8. The results are similar to MCDO and show a direct correlation between a lower λ_{reg} and a higher predicted aleatoric uncertainty. For a value of $\lambda_{reg} = 0.25$ the predicted aleatoric uncertainty is at the optimal level highlighted by the lowest aleatoric MSE in Table 4.5 and by the middle graph in Figure 4.8b. We therefore are going to use $\lambda_{reg} = 0.25$ for the following experiments.

4.3.2.3 Ensemble Size

Finally, we would like to determine how large the ensemble needs to be in order to get a good estimate of the predicted uncertainties. For that we are going to use the same hyperparameters as before but vary the number of members in the ensemble. The results for different ensemble sizes are shown in Figure 4.9 and Table 4.6. The results indicate that the predicted aleatoric uncertainty is at a similar level for all ensemble sizes, as it an average of the predicted aleatoric

Size	NLL ↓	MSE ↓	Aleatoric Error ↓	EUC ↑
4	0.709	2.625	0.008	0.783
6	0.725	2.266	<u>0.007</u>	0.770
8	0.755	2.222	0.009	0.810
10	<u>0.586</u>	1.934	<u>0.007</u>	<u>0.914</u>
15	0.599	<u>1.925</u>	0.008	0.883
20	0.631	2.150	<u>0.007</u>	0.848

Table 4.6: Performance of the DE model with different ensemble sizes: The metrics do not clearly show a trend. However, for an ensemble size of 10, the NLL, aleatoric error and EUC results are the best.

uncertainties of the individual members. The predicted epistemic uncertainty with the EUC is also at a similar level for all ensemble sizes, but with a slight increase for larger ensembles. Overall, the MSE and NLL decrease with increasing ensemble size, which is expected as the ensemble can better approximate the function by averaging over more models.

4.3.3 Repulsive Last Layer Ensembles (RLLEs)

Even though DEs has proven to deliver good results, they have two major disadvantages: On the one hand, the costs for the calculation increase significantly due to the execution of several models. On the other hand, one has to rely on the random initialization for the correct prediction of the epistemic uncertainty and thus has no guarantee that it is sufficient.

These two problems can be solved by using RLLEs. By using function space repulsion, we can force the model to make more diverse predictions in regions with little data. At the same time, applying the ensemble and repulsion only to parts of the network reduces computational effort.

For the training of RLLEs in regression task typically the Gaussian likelihood function

$$p(y|f(x;\theta)) = \frac{\|y - f(x;\theta)\|^2}{2\sigma^2} \quad (4.7)$$

with a fixed standard variance σ^2 is used [5, 25]. However, since we are also interested in the differentiation between aleatoric and epistemic uncertainty, we are replacing the Gaussian likelihood function with the heteroscedastic likelihood function as discussed in Section 2.1. The individual particles are therefore also trained to predict the variance with a second aleatoric uncertainty head. During inference the mean and variance of the ensemble is calculated by aggregating the predictions of the individual particles.

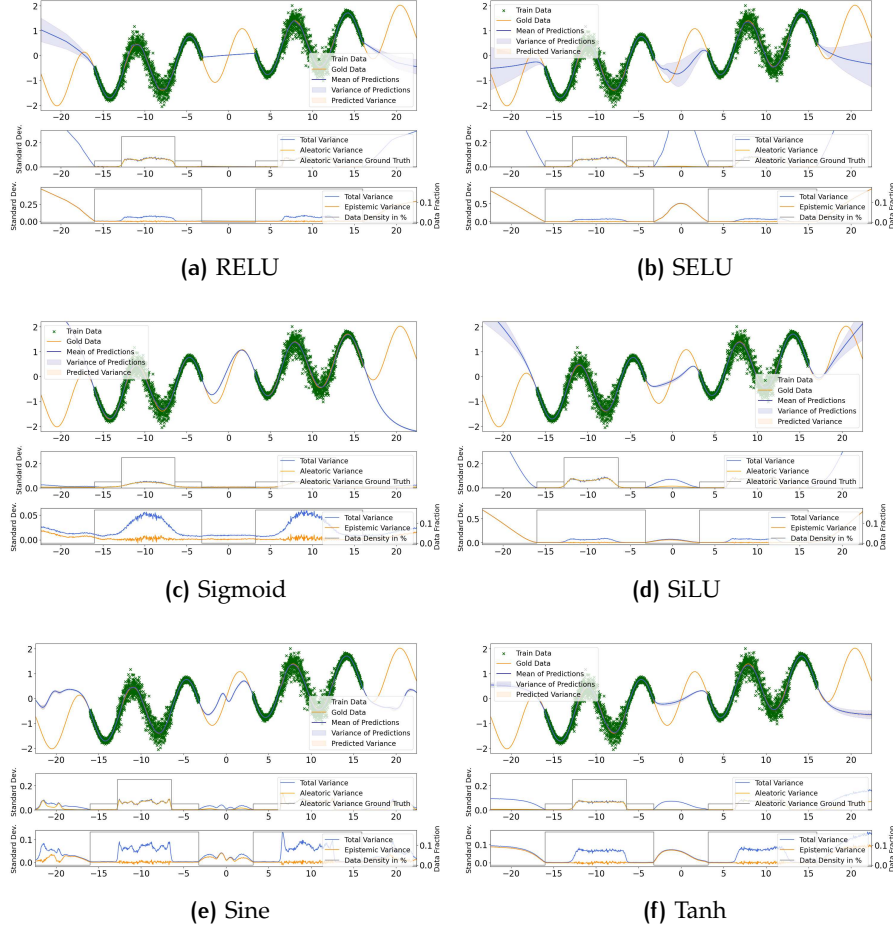


Figure 4.10: Comparison of different activation functions for RLLE. All activation functions learn from the training data and can predict aleatoric and epistemic uncertainty.

For the main model in the following experiments, we are using the same configuration as described in Section 4.3. The repulsive ensemble is also trained using a learning rate of 0.001 with ADAM for 100 epochs and has one fully connected hidden layer with 50 neurons. For the evaluation of the repulsion, we are using the range $[-20; 20]$. The variance regularizer λ is set to 1.0 and the repulsion strength γ in Equation (2.12) is also set to 1.0.

4.3.3.1 Activation Functions

Figure 4.10 shows the results for the RLLE training using different activation functions. Given the quality metrics in Table 4.7 we can see, that the SeLU activation function performs best in terms of NLL and MSE and only slightly worse than Tanh in predicting the variance.

In general, the aleatoric uncertainty prediction is at a similar level for all activation functions. A noticeable detail is that the prediction of epistemic uncertainty strongly depends on the activation function. As

Act	NLL ↓	MSE ↓	Aleatoric Error ↓	EUC ↑
RELU	0.461	0.903	0.017	0.896
SELU	0.276	0.533	0.017	0.955
Sigmoid	1.427	2.860	0.020	0.840
SiLU	0.652	1.284	0.017	0.975
Sine	0.285	0.561	0.017	0.410
Tanh	0.456	0.913	0.016	0.855

Table 4.7: Comparison of different activation functions for RLLEs: SELU achieves the best NLL and MSE results. Tanh produces the best aleatoric prediction, while SiLU is the best at predicting epistemic uncertainty.

λ_{reg}	NLL ↓	MSE ↓	Aleatoric Error ↓	EUC ↑
0.10	0.490	0.794	0.091	0.791
0.25	0.331	0.587	0.007	0.925
0.50	0.371	0.705	0.010	0.977
1.00	0.276	0.534	0.017	0.958

Table 4.8: Comparison using different variance regularizer values with RLLE: For the aleatoric uncertainty, just as with DEs, $\lambda_{reg} = 0.25$ is the best choice.

an example, by using the SELU function in Figure 4.10b, the epistemic uncertainty is high in the outer regions and in the middle, with low values in regions with data which is the desired outcome. The RELU activation function on the other hand clearly predicts a high epistemic uncertainty in the outer regions while in the middle region it predicts no epistemic uncertainty. This is most likely due to the fact that the single ensemble layer doesn't allow enough flexibility to learn complex uncertainty patterns. It should be noted that the scale and therefore absolute values of the epistemic uncertainty differ between the graphs.

As the SELU activation function can best distinguish between aleatoric and epistemic uncertainty and is fitting the function well, we will use it for the following experiments.

4.3.3.2 Variance Regularizer

The results for different values of the variance regularizer are shown in Figure 4.11 and Table 4.8. They show a direct correlation between a lower λ_{reg} and a higher predicted variance, similar to MCDO and DEs. For a λ_{reg} of 0.25 the predicted variance is closed to the true variance, which can be seen by the low aleatoric error in Table 4.8.

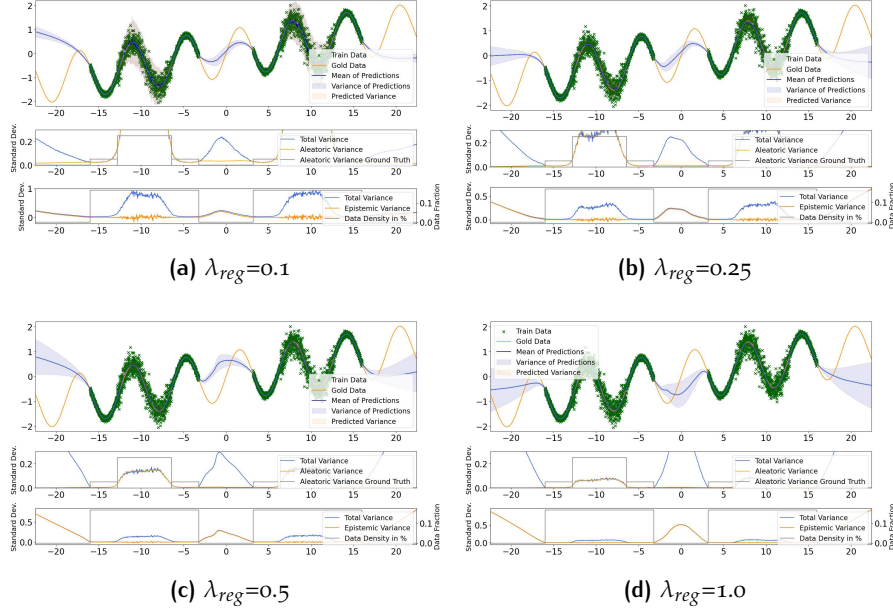


Figure 4.11: Comparison of different variance regularizer values using RLLEs: When λ_{reg} is set to 0.25, the prediction of aleatoric uncertainty is the most accurate, as it is only slightly overestimated.

4.3.3.3 Calibration of Repulsion Strength

Even if we get excellent results with SELU as the activation function, it may be necessary to use a predefined activation function for certain applications. This can especially be the case if an existing model is to be extended by adding a RLLE. Since optimizing all hyperparameters is often computationally intensive, it would be beneficial to reuse as many as possible.

To show this with an example, we re-examine the case using the Sine activation function from Figure 4.10e. In Figure 4.12 we show the result of the repulsive last layer ensemble with an already calibrated λ_{reg} of 0.25. The results show that the RLLE is able to predict the aleatoric uncertainty well, but the epistemic uncertainty is lower than expected.

As described in Section 2.4 the repulsion is realized by

$$f_i^{t+1} \leftarrow f_i^t + \epsilon_t \phi(f_i^t) \quad (4.8)$$

with $\phi(f_i^t) = \nabla_{f_i^t} \log p(f_i^t | \mathcal{D}) - \gamma \mathcal{R}(\{\nabla_{f_i^t} k(f_i^t, f_j^t)\}_{j=1}^n)$

where $\nabla_{f_i^t} \log p(f_i^t | \mathcal{D})$ is driving the ensemble members towards the desired function and $\mathcal{R}(\{\nabla_{f_i^t} k(f_i^t, f_j^t)\}_{j=1}^n)$ is the repulsion term that is enforcing diverse predictions. We can now use the repulsion strength γ to control the impact of the repulsion term on the overall loss. A low value for γ means that the repulsion term has a low impact on the loss and therefore the ensemble members are not as strongly forced to make diverse predictions. A high value for γ means that the repulsion

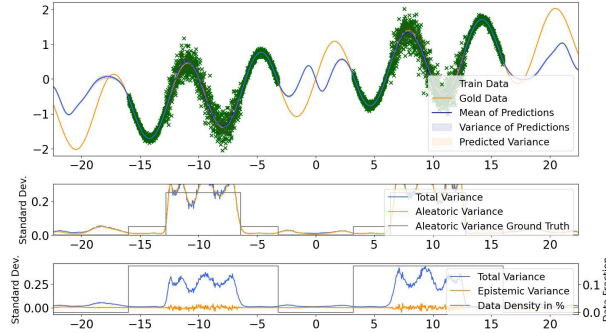


Figure 4.12: Result for the Sine activation function using RLLE with $\lambda_{reg} = 0.25$ and $\gamma = 1.0$. The predicted aleatoric uncertainty is at a good level, but the epistemic uncertainty is not at the desired level.

γ	NLL ↓	MSE ↓	Aleatoric Error ↓	EUC ↑
0.1	0.159	0.234	<u>0.009</u>	0.142
1	0.159	0.234	<u>0.009</u>	0.142
100	0.159	0.234	<u>0.009</u>	0.150
1000	<u>0.151</u>	<u>0.216</u>	0.010	0.875
5000	0.173	0.260	0.010	<u>0.895</u>
10000	0.614	1.005	0.010	0.696

Table 4.9: Results of the RLLE model using the Sine activation function with different values of repulsion strengths γ . The selection of $\gamma = 5000$ results in a significant enhancement of the epistemic uncertainty prediction in comparison with $\gamma = 1$.

term has a higher impact on the loss and therefore the ensemble members make more diverse predictions. Increasing the γ should therefore increase the epistemic uncertainty.

Varying the repulsion strength γ produces the results shown in Figure 4.13 and Table 4.9. For $\gamma \leq 100$ the results stay nearly unchanged, with only a marginal increase for the EUC. For $\gamma = 1000$ and $\gamma = 5000$ the predicted epistemic uncertainty significantly improves, where the former has a slightly better NLL and MSE and the latter a better EUC. Both values result in a minor decline in the predicted aleatoric uncertainty, but the values are still at a good level. Using $\gamma = 10000$ the performance of the predicted epistemic uncertainty and even the mean prediction decrease significantly.

These results show us that by choosing the correct γ , we can also use activation functions to predict uncertainties that previously did not produce good results. Together with λ_{reg} we can therefore influence both the predicted aleatoric and epistemic uncertainty to obtain optimal results.

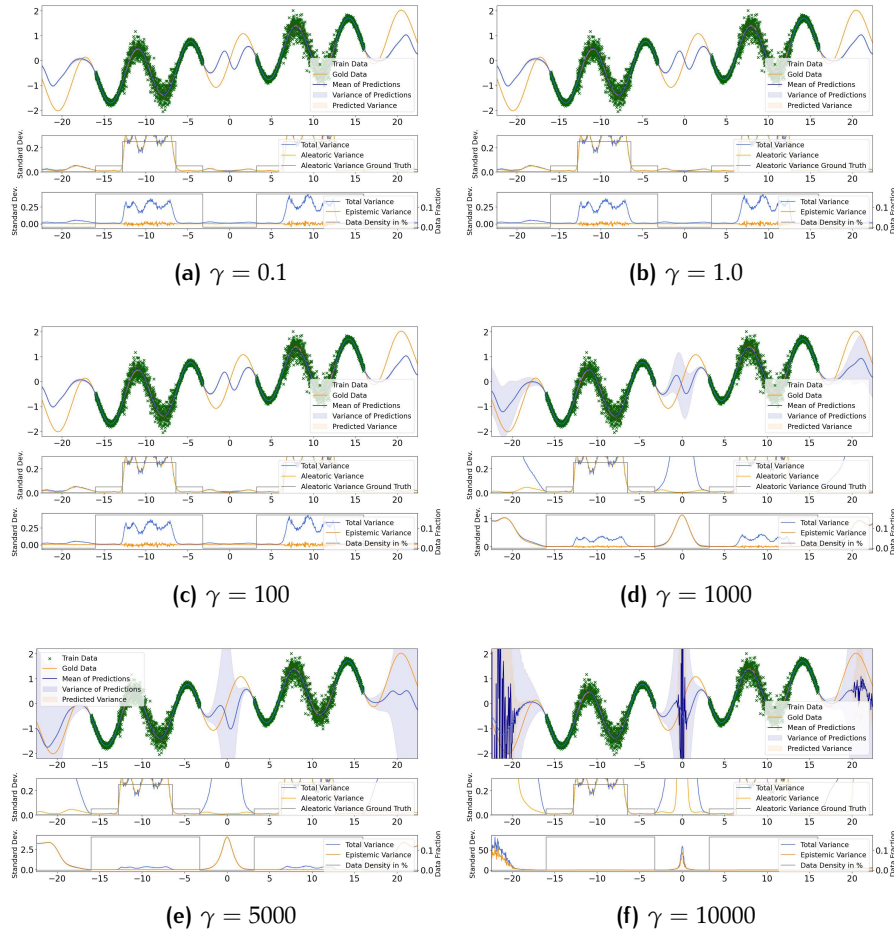


Figure 4.13: Comparison of different gamma values using RLLEs: The larger γ is chosen, the more epistemic uncertainty is predicted, without significant influence on the mean or aleatoric uncertainty predictions.

Method	NLL ↓	MSE ↓	Aleatoric Error ↓	EUC ↑
MCDO (Dropout: 0.1)	0.708	1.385	0.008	0.640
MCDO (Dropout: 0.15)	0.169	0.278	0.007	0.241
DE	0.586	1.934	0.007	0.914
RLLE	0.331	0.587	0.007	0.925

Table 4.10: Performance Comparison between the different methods. The aleatoric error is similar for all shown approaches. MCDO with a dropout rate of 15% has the best NLL and MSE, but has a significantly worse EUC compared to the other results. RLLE has the second best NLL and MSE with the best EUC and performs the best overall.

4.4 CONCLUSION

In this chapter, we have explored the application of MCDO, DEs, and RLLEs for uncertainty estimation of a simple regression task using the Noisy Sine dataset. We calibrated different hyperparameters for each approach to obtain the best possible results. For all methods, we first find a suitable activation function, then adjust the variance regularizer to minimize the aleatoric error and finally calibrate the method specific hyperparameters.

Table 4.10 shows the best results for each method to allow a direct comparison. It is evident that all methods are capable of fitting the training data and are predicting aleatoric uncertainty with reasonable accuracy. However, the model fitting and especially the prediction of epistemic uncertainty varies significantly depending on the method and hyperparameter choices.

As an example, MCDO with a dropout rate of 0.15 fits the data well but struggles to predict epistemic uncertainty. Reducing the dropout rate to 0.1 leads to a better prediction of epistemic uncertainty, but the overall performance drops significantly.

DEs show a robust performance with a good balance between aleatoric and epistemic uncertainty prediction. However, they solely rely on random initialization to achieve diversity with only the ensemble size as a tunable hyperparameter. RLLEs, on the other hand, provided a more flexibility in terms of hyperparameter tuning, as the repulsion strength γ directly impacts the diversification of the ensemble members. This allows the epistemic uncertainty to be controlled, which requires additional parameter optimization, but improves the results.

Overall, RLLEs provide the best results, offering a good balance between predictive performance and uncertainty estimation. The results highlight in particular the importance of hyperparameter optimization for all methods when applying BNNs to regression tasks.

5

BAYESIAN NEURAL NETWORKS FOR DEPTH ESTIMATION

In Chapter 4 we have seen that the various BNN approaches, although of varying degrees of success, could provide uncertainty estimates for a synthetic toy data set. However, the aim of developing BNNs should not only be of a theoretical nature, but the objective should always be to apply them in real-world applications.

The target of this chapter is to apply the discussed BNN approaches to the task of monocular depth estimation. We will first introduce the task itself and the dataset used. Afterwards, the architecture and training procedure of the model will be described. Finally, we will evaluate the predictive and runtime performance of the different approaches and compare them with each other.

5.1 MONOCULAR DEPTH ESTIMATION

Depth estimation is the task of determining depth information of a scene from given camera data [8]. There are two main approaches to this task: stereo depth estimation and monocular depth estimation. Stereo depth estimation uses two cameras to estimate the depth of a scene, while monocular depth estimation uses just a single camera [15]. In this work, we will focus on monocular depth estimation, although the stereo approach works almost identically.

In monocular depth estimation, the model receives a single RGB image as an input and is expected to output a depth map, where it predicts the distance from the camera for each pixel in the image. We therefore have an input image of the size $H \times W \times 3$, where H and W are the height and width of the image with 3 color channels (RGB). The output depth map has the size $H \times W \times 1$, where each pixel in the depth map is a scalar representing the distance of the corresponding pixel in the input image. The task for the model is to learn a mapping from the input image to the output depth map.

In comparison to the previous toy dataset in Chapter 4, this regression task is much more complex. This is mainly because in the previous task both the input and the output were one-dimensional. With the monocular depth estimation, however, we have a high-dimensional input and output. In comparison to stereo depth estimation, where we have two distinct images and usually know the relative positions between the cameras, we can not use triangulation to estimate the

depth [16]. Instead, we have to solely rely on the information contained in a single input image.

5.1.1 Dataset: KITTI

The KITTI dataset [13, 32] is a widely used dataset for training depth estimation, but can also be used for optical flow, odometry or object detection. It contains multiple scenes with about 40000 stereo image pairs, that were taken by driving a car around the city of Karlsruhe, Germany. The images are taken by two cameras mounted on the car and have a resolution of mostly 1242×375 pixels.

In addition to these images, the dataset also contains corresponding depth maps, which were collected by a LiDAR sensor mounted on the car. As an example, we show an image from the KITTI dataset and scanned depth map from the LiDAR in Figure 5.1. Since a LiDAR sensor was used to collect the depth maps, they are accurate but also sparse. This means that only around 4% of all pixels of the input image have a valid depth value. The sparse data can also be observed in Figure 5.1b, although the data is already enhanced to improve its visibility. As the upper part of the image never contains depth information, we could also crop the image to a smaller size raising the percentage of valid pixels to about 11.5%.

This sparsity of ground truth data makes the task of learning monocular depth estimation even more difficult. A solution would be to capture more images with a LiDAR sensor which is not feasible in practice as it requires a lot of resources and time. To overcome the problem of sparse depth maps, we can also use a self-supervised learning approach. This means that we do not need to have a dense ground truth depth map for every image. Instead, we can use the temporal connection between the images to extract information about the distance of objects.

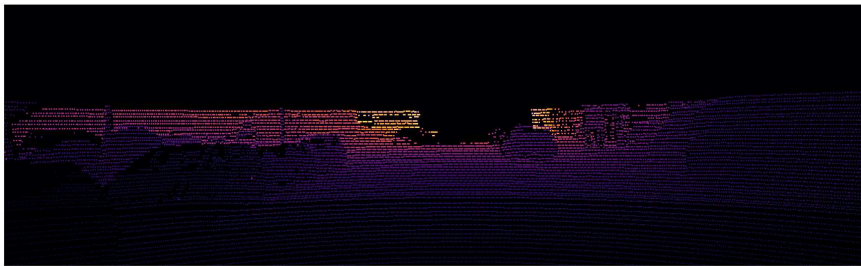
5.1.2 Neural Architecture: Monodepth 2

There are a variety of approaches and neural network architectures that can be used for depth estimation. From rather simpler convolutional neural networks [8], to more specialized architectures [10], to joining the trend in recent years to use apply vision transformer architectures for depth estimation [3, 45].

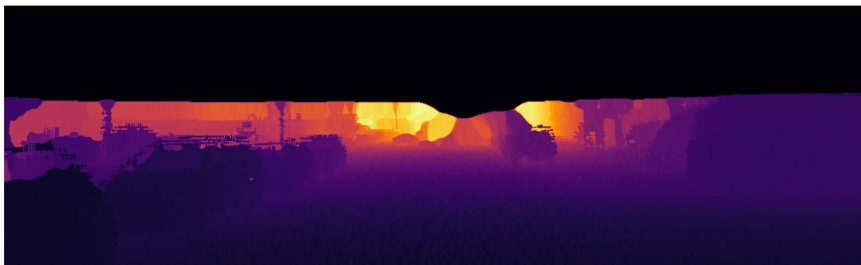
While vision transformer architectures currently deliver the best results, they also have disadvantages: In some cases, only the trained weights of the models are given and the hyperparameters for the training process are not provided. This makes it difficult to reproduce the results, and prevents the retraining of networks, which is necessary for many BNNs.



(a) Example image from KITTI dataset



(b) Corresponding sparse LiDAR depth map



(c) Enhanced visualization of the depth map

Figure 5.1: Visualization of the KITTI dataset [32]: (a) Example image from the KITTI dataset with corresponding depth map. (b) The depth map is sparse, meaning that only a few pixels have a valid depth value. (c) For a better visualization missing depth values are interpolated.

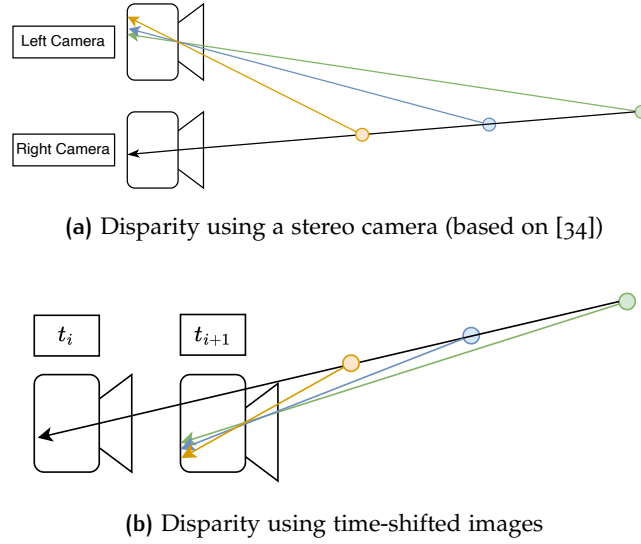


Figure 5.2: Visualization of disparity for a stereo camera (a) and time-shifted images (b). The disparity of the three objects (orange, blue, and green circle) is dependent on the distance from the camera.

Most importantly, however, BNNs always generate an overhead of computational cost that can not be neglected. Depending on the concrete implementation, this can be significant and result in much longer training times. In addition, vision transformers are often much larger in terms of the number of parameters and require much more training data to achieve their full potential. This combination would lead to extremely long training times.

As a compromise between good performance and trainability, we use Monodepth 2, which is primarily based on convolutional layers [16]. This decision is also supported by the fact that Monodepth 2 is only trained using the KITTI dataset and provides all parameters necessary to reproduce the results from the original paper [16].

Monodepth 2 is a self-supervised learning approach, that does not require the ground truth data from KITTI for the training process. Instead, it learns by predicting the depth information from one image and checking its accuracy against a second image. It applies multiple strategies to improve the depth estimation performance, which we will cover in the following.

The self-supervised learning approach of Monodepth 2 is based on the concept that we can use any connection between two or more images to estimate the depth in a scene. For stereo images, where the same scene is captured simultaneously from two different angles, we can use the disparity between the two images to estimate the depth. In this case, disparity refers to the difference in position of the same object in the two images.

For a better understanding, we have illustrated the concept of disparity with a simple example in Figure 5.2a. In the image two cameras,

which are positioned next to each other, and three objects (orange, blue, and green circle) are shown. All three objects are projected to the same pixel in the image of the right camera. On the left camera, however, the objects are projected to different pixels depending on the distance to the cameras. The closer an object is to the camera, the larger the disparity between the camera gets. In the example, the green circle is far away from the camera, so the disparity between the two images is small. The orange circle, on the other hand, is very close to the camera, so the disparity is larger.

The disparity and depth are therefore inversely proportional to each other. To calculate the *depth* from the *disparity*, we can use

$$depth = \frac{bf}{disparity} \quad (5.1)$$

where b is the baseline distance between the two cameras and f is the focal length of the camera [15]. The focal length is a property of the camera and is usually known. The distance between the two cameras in a stereo setup is usually also known as the cameras are mounted on a fixed rig. This inversion of disparity to depth can theoretically have an effect on the probability distribution [26]. However, we are primarily interested in the uncertainty of the pixels and therefore use the probability distribution of the disparity.

For the case of a monocular camera, we can use the same concept, but instead of using two cameras, we use two images taken at different times and reposition the camera between them. As the visualization in fig. 5.2b shows, this produces a similar result to before. One major downside of this approach is that the distance between the two images is not known, as it depends on multiple factors. In the used KITTI dataset, the car can be moving at different speeds between two images, can be turning, or even standing still.

To estimate the position of the camera between the two frames, Monodepth 2 uses a neural network called PoseNet that estimates the relative change of position between frames. The two frames are provided to the PoseNet, it forwards each image through a ResNet [18], combines the output features of the ResNet using a smaller convolutional network, and thereby predicts the rotation and translation between the frames.

The main neural network Monodepth 2 is based on is a Fully Convolutional Networks (FCN) [30], that uses an Encoder-Decoder-Architecture for the depth prediction. FCNs are solely based on convolutional layers and do not use any fully connected layers. This allows them to process images of arbitrary size and require a lower number of parameters than fully connected networks. This is especially relevant due to the large input and output dimensions. A classification network, for example, only has n output neurons, where n is the number of classes. In our case we need $H \times W$ output neurons, depending on the input image height H and width W .

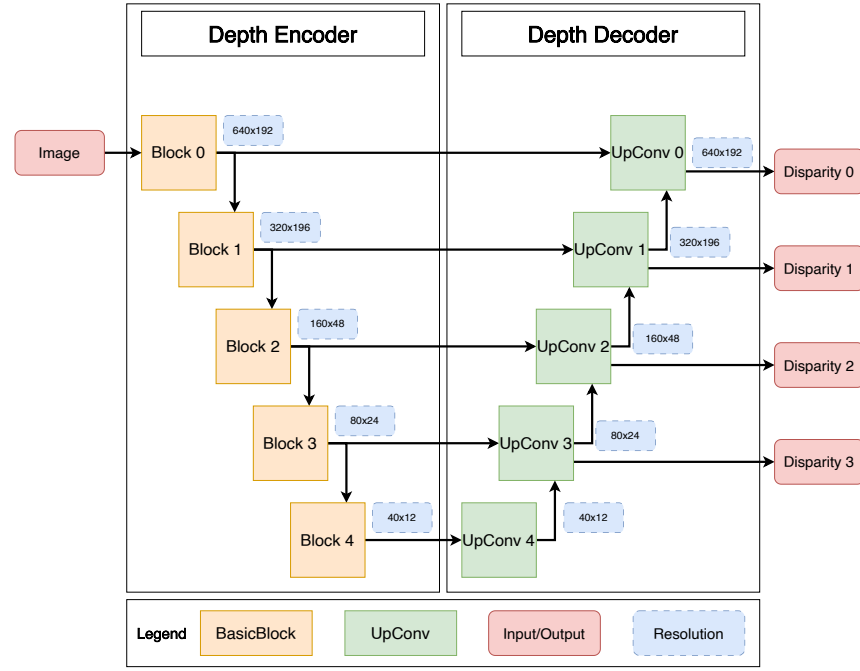


Figure 5.3: U-Net [37] Architecture of the depth estimation network of Monodepth 2 [16]. The ResNet encoder-blocks forward their results to the corresponding UpConv block of the decoder.

The architecture of Monodepth can be seen in Figure 5.3 with the encoder on the left and the decoder on the right. The architecture is based on the U-Net [37] architecture, which was originally proposed for image segmentation tasks, but is widely used for other tasks. The U-Net architecture consists of an encoder and a decoder, where the encoder extracts features from the input image and the decoder reconstructs the output from these features. The intermediate features of each encoder layer are forwarded to the corresponding decoder layers. The deeper layers, in the example Figure 5.3 the blocks on the bottom, therefore produce more coarse and abstract features which get refined in the shallower layers.

The encoder is based on the ResNet18 [18], which consists of multiple basic blocks that include convolutional layers with residual connections. For the details, we refer the reader to the original paper [18]. In order to increase performance, the ResNet is initialized to a pre-trained model on ImageNet [38].

The decoder consists of multiple Upscale-Convolutional-Block that apply convolutional layers to the features and up-sample them to increase the resolution. In Figure 5.4 the structure of these blocks is shown: A Upscale-Convolutional-Block (UpConv-Block) receives the features from the block below, with an exception for the deepest block. A convolutional layer and the activation function is applied, and the output are upsampled by interpolating, which doubles the height

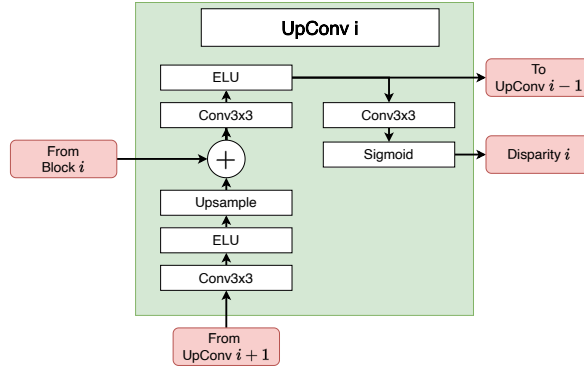


Figure 5.4: The UpConv block of the Monodepth 2 architecture receives the result of the previous UpConv block and of the corresponding encoder block. The output is transferred to the subsequent UpConv block, and an intermediate disparity map is produced.

and the width of the features. Then the residual connection from the corresponding encoder block is added. After another convolutional layer, the features forwarded to the next block.

Additionally, each UpConv-Block produces an intermediate disparity map with its according resolution. By calculating and applying gradients using this intermediate results, the model avoids getting stuck in local minima due to the bilinear up-sampling process [16, 22]. Regarding the activation functions, the encoder uses RELU, the decoder uses ELU, while Sigmoid is used for output activation.

Using the outputs of the PoseNet and the predicted depth of the depth network the loss can then be calculated. The loss function of Monodepth 2 is based on novel view-synthesis, which is a technique of predicting an image J from a different viewpoint [16]. By using the predicted disparity of given image $\hat{I}$ and the predicted pose change between the images $\hat{I}$ and $\hat{J}$, Monodepth 2 generates a predicted image J . This is done by creating a point cloud from the depth information and projecting this point cloud to the image plane based on the pose change. The closer the generated image J is to the original image $\hat{J}$, the better the depth estimation and pose estimation is [16].

The loss function of Monodepth 2 is based on two components: The photometric error, which is minimizing the difference between the generated image J and the original image $\hat{J}$ and the smoothness loss, which is ensuring a smoother prediction by reducing the difference between depth values of adjacent pixels. Additionally, a mask is used to ignore pixels where a calculation of loss values is not effective. These are mainly cars driving ahead and are not changing their distance relative to the camera because they are moving at the same speed as the camera [16].

Metric	Formula
Absolute Relative Diff. (Abs Rel)	$\frac{1}{N} \sum_{i=1}^N \frac{ y_i - \hat{y}_i }{\hat{y}_i}$
Root Mean Squared Error (RMSE)	$\sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2}$
Accuracy: % of y_i with $\delta < thr$	$\max(\frac{y_i}{\hat{y}_i}, \frac{\hat{y}_i}{y_i}) < thr$

Table 5.1: Evaluation metrics for depth estimation proposed by Eigen et al. [8]

Abs Rel	RMSE	$\delta < 1.25$
0.1166	4.8560	0.8725

Table 5.2: Results of the depth estimation performance evaluation of Monodepth 2 on the KITTI dataset

5.1.3 Evaluation Metrics

The evaluation of the different BNN approaches is done using the KITTI dataset which was introduced in Section 5.1.1. For the evaluation of the performance, we use the data split proposed by Eigen et al. [8] to allow for a fair comparison with other approaches. Unless otherwise stated, we use the original training parameters of Monodepth 2 to achieve the best possible results. The training is done using the Adam optimizer with a learning rate of 0.0001 for 20 epochs and a batch size of 12. To reduce the computational cost, we use a smaller image resolution of 192×640 pixels.

The evaluation of the mean predictions is conducted using the error metrics widely used by other research papers [8, 16, 45]. The equations used for the calculation are shown in Table 5.1. All metrics are calculated using the predicted depth map y and the ground truth depth map $\hat{y}$ masked to all valid depth values of the ground truth depth map.

The different metrics provide an overview of the performance from different perspectives. The Absolute Relative Difference is especially sensitive for regions where the ground truth data is small. In other words, errors for close objects are more important than errors for objects that are further away. The Root Mean Squared Error on the other hand averages without taking the distance into account. The accuracy tells which percentage of the predicted pixels are close to the ground truth data. The value for the threshold is usually set to 1.25. The performance for the depth prediction of Monodepth 2 can be seen in Table 5.2.

To evaluate the uncertainty prediction of the different approaches we use two metrics: The Area Under the Sparsification Error [21, 36] indicates whether the predicted uncertainty is in line with the error of a prediction. This is done by creating a sparsification plot, that is

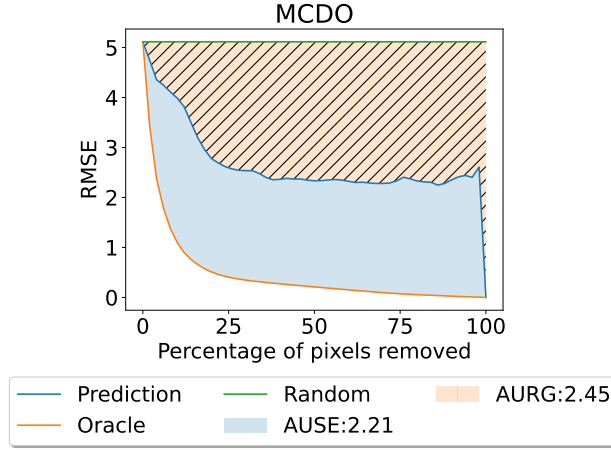


Figure 5.5: Visualization of the calculation of Area Under the Sparsification Error (AUSE) and Area Under the Random Gain (AURG). The AUSE is defined by the area between the prediction (blue graph) and the oracle (orange graph), and the AURG is the area between the prediction (blue graph) and random sparsification (green graph).

obtained by ordering the pixels by their predicted uncertainty. An ever-increasing percentage of these pixels starting with the highest uncertainty is then repeatedly removed, and a metric like the Root Mean Squared Error (RMSE) is calculated. Additionally, a sparsification is also done for an oracle, by ordering the pixels depending on their true error. The Area Under the Sparsification Error (AUSE) is then calculated by computing the area between the oracle curve and the predicted uncertainty curve [20].

The Area Under the Random Gain [36] works similarly, but instead of the oracle we use a random sparsification to evaluate if the model is better than guessing. Figure 5.5 illustrates the calculation of the AUSE and AURG for a MCDO prediction. The AUSE and AURG are calculated using RMSE, Absolute Relative Difference (Abs Rel) and accuracy as base metrics.

5.2 MONTE CARLO DROPOUT (MCDO)

Analogous to Chapter 4 we are going to start with the evaluation of MCDO [11]. In contrast to before, however, we only have the dropout rate that we are varying because all other hyperparameters are already given by the model. In order to allow the model to learn the uncertainty, we apply the dropout with the according dropout rate during the training and evaluation. The 2D dropout is applied to the decoder part of the model. The dropout is therefore applied to whole feature maps and not to each pixel individually.

The results for the depth prediction of MCDO can be seen in Table 5.3 and the results for the predicted uncertainty in Table 5.4. The result

Dropout Rate	Abs Rel ↓	RMSE ↓	$\delta < 1.25$ ↑
0.001	<u>0.1170</u>	4.8481	<u>0.8735</u>
0.002	0.1175	4.8643	0.8718
0.003	0.1191	4.8879	0.8706
0.004	0.1184	4.8920	0.8702
0.005	0.1212	4.9311	0.8678
0.006	0.1200	4.9150	0.8702
0.007	0.1187	4.9116	0.8718
0.008	0.1181	4.8588	0.8715
0.009	0.1197	4.8523	0.8690
0.010	0.1185	4.8549	0.8694
0.020	0.1185	4.8477	0.8697
0.030	0.1205	4.8645	0.8645
0.040	0.1195	4.8504	0.8654
0.050	0.1205	4.8452	0.8626
0.060	0.1220	4.8906	0.8634
0.070	0.1215	4.8694	0.8622
0.080	0.1224	4.8919	0.8603
0.090	0.1234	4.8966	0.8603
0.100	0.1228	4.8212	0.8576
0.150	0.1229	<u>4.7925</u>	0.8569
0.200	0.1252	4.8981	0.8508
0.300	0.1302	4.8155	0.8445
0.400	0.1309	4.9107	0.8390
0.500	0.1362	5.0628	0.8266

Table 5.3: Depth Prediction Performance of MCDO using Monodepth 2 with different dropout rates: The results of the Abs Rel and accuracy show, that increasing the dropout rate directly harms the depth estimation performance.

Dropout Rate	Abs Rel		RMSE		$\delta < 1.25$	
	AUSE ↓	AURG ↑	AUSE ↓	AURG ↑	AUSE ↓	AURG ↑
0.001	0.0564	0.0197	3.1036	1.0122	0.0742	0.0409
0.002	0.0494	0.0271	2.8783	1.2497	0.0628	0.0539
0.003	0.0525	0.0251	3.0965	1.0472	0.0672	0.0505
0.004	0.0474	0.0299	2.8211	1.3302	0.0596	0.0584
0.005	0.0457	0.0331	2.4352	1.7402	0.0558	0.0639
0.006	0.0487	0.0296	2.9326	1.2353	0.0601	0.0579
0.007	0.0456	0.0314	2.5312	1.6353	0.0542	0.0623
0.008	0.0450	0.0320	2.4916	1.6304	0.0549	0.0620
0.009	0.0452	0.0318	2.5157	1.5829	0.0555	0.0631
0.010	0.0428	0.0343	2.1895	1.9223	0.0511	0.0676
0.020	0.0455	0.0312	2.1085	1.9990	0.0557	0.0625
0.030	0.0480	0.0298	2.1314	1.9751	0.0622	0.0602
0.040	0.0487	0.0285	1.9407	2.1595	0.0629	0.0586
0.050	0.0523	0.0252	2.0195	2.0683	0.0726	0.0514
0.060	0.0544	0.0251	2.0893	2.0458	0.0732	0.0501
0.070	0.0574	0.0214	2.2298	1.8824	0.0819	0.0422
0.080	0.0595	0.0197	2.2252	1.9043	0.0855	0.0404
0.090	0.0606	0.0193	2.2830	1.8473	0.0863	0.0394
0.100	0.0613	0.0180	2.3288	1.7272	0.0912	0.0364
0.150	0.0700	0.0098	2.5918	1.4389	0.1082	0.0203
0.200	0.0721	0.0089	2.5529	1.5692	0.1133	0.0197
0.300	0.0846	-0.0008	2.7483	1.2648	0.1339	0.0036
0.400	0.0868	-0.0022	3.0119	1.0895	0.1426	-0.0002
0.500	0.0885	-0.0013	3.6130	0.6022	0.1532	-0.0010

Table 5.4: Uncertainty Prediction Performance of MCDO using Monodepth2 with different dropout rates: For a dropout rate of 1% the Abs Rel and accuracy metrics are best, while for 4% the RMSE based ones are best.

of the depth prediction performance clearly shows a negative impact of the dropout rate on the performance. The higher the dropout rate is chosen, the worse the performance metrics get. This is an expected effect, as the dropout adds noise to the neuron outputs and thereby also adds noise to the model prediction. The best prediction of the mean depth estimation is achieved with a dropout rate of 0.1%.

When looking at the uncertainty of the predicted we can see the best performance is reached at a dropout rate of 1% for all metrics except the AUSE of the RMSE where a dropout rate of 4% performs better. Unfortunately, it is not easy to interpret the metrics in this case, which is why additional example images are shown in Figure 5.6a and Figure 5.6b. Each figure illustrates the input image, the predicted mean and standard deviation of the disparity map. The mean is the predicted disparity map, while the standard deviation is the predicted uncertainty. The images are calculated with a dropout rate of 1% and 10%.

We can see that a dropout rate of 1% performs better than 10% in terms of the mean prediction, which is expected given the results in Table 5.3. With a dropout rate of 10% we can see that some details in the depth maps, like sharper edges, are lost.

In comparison, the uncertainty of the predicted disparity map is also different between the two dropout rates. Using a dropout of 1% has a tendency of predicting a high uncertainty on sharp edges. This is useful in so far as it may not be possible to determine the exact position of the depth change for an edge. Unfortunately, there are currently no standards regarding the areas in which the uncertainty should actually be high, which is why we have to rely on intuitive estimates. Setting the dropout to 10% results in an overall higher uncertainty prediction across the whole image, which is counterproductive as the important uncertainties might get obscured.

5.3 DEEP ENSEMBLES (DES)

We evaluate the DEs [25] in the same way as MCDO, except that we vary the size of the ensemble instead of the dropout rate. The training of DEs is comparatively expensive compared to MCDO, as not just one model, but many have to be trained. Therefore, we want to find out how many ensemble members are needed to obtain a reliable prediction of the uncertainty. Compared to MCDO, the training process is even simpler, as the ensemble is based on M individual models trained in the conventional way using different seeds.

Table 5.5 demonstrates that the depth prediction performance of the ensemble is even better than the performance of the base model in Table 5.2. In fact, the performance is increasing across all metrics the larger the ensemble gets. This is most likely due to the fact that

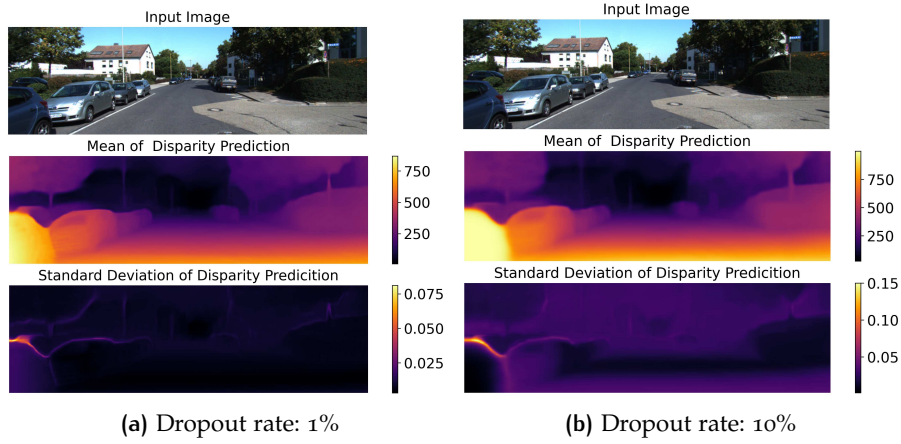


Figure 5.6: Comparison of predicted mean and variance disparity map with a dropout rate of 1% and 10%. With the increased dropout rate the edges of the depth prediction become less sharp, and the overall predicted uncertainty increases.

Size	Abs Rel ↓	RMSE ↓	$\delta < 1.25$ ↑
2	0.1132	4.7256	0.8787
3	0.1119	4.6862	0.8804
4	0.1117	4.6704	0.8804
5	0.1119	4.6626	0.8806
6	0.1116	4.6497	0.8808
7	0.1112	4.6435	0.8811
8	0.1111	4.6366	0.8815
9	0.1110	4.6340	0.8816
10	0.1109	4.6336	0.8815
11	0.1108	4.6337	0.8816
12	0.1108	4.6316	0.8818
13	0.1107	4.6302	0.8820
14	0.1109	4.6298	0.8820
15	0.1108	4.6273	0.8821
16	0.1108	4.6278	0.8822

Table 5.5: Depth Prediction Performance of DEs with different ensemble sizes: As the number of ensemble members increases, the depth prediction tends to improve.

Size	Abs Rel		RMSE		$\delta < 1.25$	
	AUSE ↓	AURG ↑	AUSE ↓	AURG ↑	AUSE ↓	AURG ↑
2	0.0605	0.0135	3.6848	<u>0.3330</u>	0.0822	0.0290
3	0.0615	0.0113	3.8320	0.1510	0.0833	0.0264
4	0.0582	0.0143	3.6752	0.2924	0.0762	0.0334
5	0.0571	0.0155	3.6515	0.3067	0.0741	0.0354
6	0.0579	0.0145	3.7096	0.2369	0.0755	0.0337
7	0.0563	0.0157	3.6531	0.2883	0.0724	0.0365
8	0.0564	0.0156	3.6795	0.2562	0.0726	0.0360
9	<u>0.0559</u>	<u>0.0160</u>	<u>3.6471</u>	0.2863	0.0715	0.0371
10	<u>0.0559</u>	<u>0.0160</u>	3.6709	0.2627	0.0716	0.0370
11	0.0572	0.0146	3.7736	0.1605	0.0736	0.0349
12	0.0565	0.0153	3.7344	0.1981	0.0719	0.0364
13	0.0560	0.0157	3.7009	0.2304	<u>0.0707</u>	<u>0.0375</u>
14	0.0564	0.0154	3.7399	0.1901	0.0713	0.0369
15	0.0594	0.0124	3.9405	-0.0126	0.0771	0.0310
16	0.0588	0.0129	3.9038	0.0248	0.0760	0.0320

Table 5.6: Uncertainty Prediction Performance of DEs with different ensemble sizes: When the ensemble size is set at 9, the Abs Rel metrics and the AUSE of the RMSE are shown to be optimal. With $n = 13$, the accuracy based metrics are optimal.

Size	Abs Rel ↓	RMSE ↓	$\delta < 1.25$ ↑
2	0.1299	5.0646	0.8491
3	0.1295	5.0100	0.8490
4	0.1288	4.9934	0.8505
5	0.1283	4.9767	0.8511
6	0.1280	4.9706	0.8517
7	0.1279	4.9689	0.8519
8	0.1278	4.9701	0.8522
9	0.1274	4.9552	0.8528
10	0.1272	4.9540	0.8531
11	0.1269	4.9479	0.8535
12	0.1269	4.9439	0.8537
13	0.1266	4.9384	0.8542
14	0.1265	4.9363	0.8542
15	<u>0.1264</u>	4.9327	<u>0.8545</u>
16	<u>0.1264</u>	<u>4.9305</u>	0.8544

Table 5.7: Depth Prediction Performance of a randomly initialized DE with different ensemble sizes: In this instance, a larger ensemble has also produces better depth predictions.

the ensemble is able to learn more diverse features and therefore can generalize better.

The performance of the uncertainty prediction for the DE is shown in Table 5.6. We can see that the DE seems to have an optimum for the ensemble size, that results in the best uncertainty prediction performance, but overall performance difference is rather small. Depending on the exact metric the optimal ensemble size is around 9. In order to predict an output distribution, the ensemble needs a certain size, which is why the performance is increasing with the ensemble size. However, if the ensemble size is too large, the predictions seem to become too similar and result in too confident predictions.

The core concept on which DEs are based is the random initialization of the ensemble members. Usually, the depth encoder of Monodepth 2 is initialized with pre-trained weights from a ImageNet. By omitting this initialization and using random weight values for the encoder instead, we can increase the possible parameter space.

By comparing the results of Table 5.5 and Table 5.7 we can see that the depth prediction is worse with randomly initialized weights, which is expected as the depth prediction performance of each individual ensemble member gets worse. Comparing the outputs of these two approaches in Figure 5.7 we observe that the visual difference of the depth prediction is very subtle.

Size	Abs Rel		RMSE		$\delta < 1.25$	
	AUSE ↓	AURG ↑	AUSE ↓	AURG ↑	AUSE ↓	AURG ↑
2	0.0639	0.0197	3.6638	0.5837	0.0914	0.0425
3	0.0609	0.0219	3.5965	0.5945	0.0839	0.0500
4	0.0583	0.0241	3.5144	0.6645	0.0771	0.0558
5	0.0569	0.0252	3.4339	0.7315	0.0732	0.0592
6	0.0567	0.0252	3.4280	0.7325	0.0722	0.0597
7	0.0557	0.0263	3.3817	0.7786	0.0697	0.0621
8	0.0556	0.0262	3.3740	0.7875	0.0693	0.0622
9	0.0548	0.0269	3.3344	0.8154	0.0676	0.0634
10	0.0542	0.0273	3.2899	0.8598	0.0664	0.0644
11	0.0538	0.0275	3.2714	0.8738	0.0656	0.0648
12	0.0563	0.0250	3.4783	0.6636	0.0701	0.0601
13	0.0563	0.0248	3.4844	0.6532	0.0705	0.0594
14	0.0558	0.0252	3.4496	0.6863	0.0695	0.0603
15	0.0555	0.0254	3.4326	0.7008	0.0690	0.0607
16	0.0551	0.0259	3.3991	0.7319	0.0681	0.0616

Table 5.8: Uncertainty Prediction Performance of a randomly initialized DE with different ensemble sizes: When the ensemble size is set to 13, the performance of all metrics is optimal.

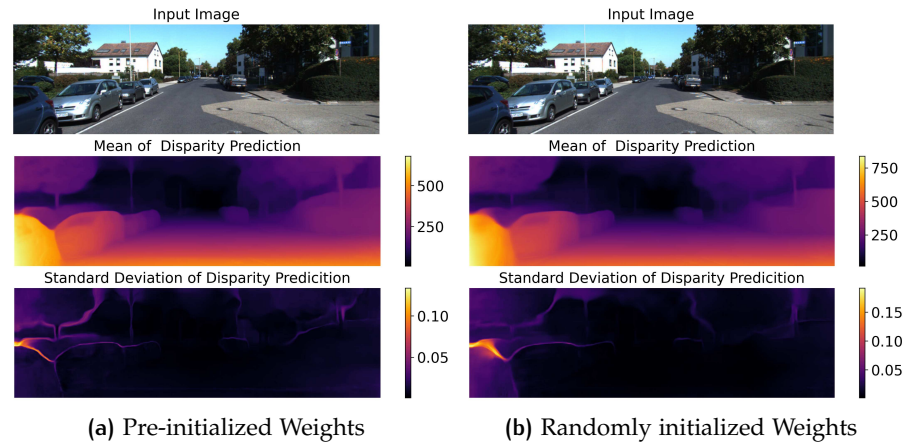


Figure 5.7: Comparison of the depth prediction performance of DE with pre-trained weights (a) and randomly initialized weights (b). The predictions of depth and uncertainty are similar, but that the edges are clearer with the pre-initialised DE.

Comparing the uncertainty prediction in Table 5.6 and Table 5.8 we can see that the performance seems to be better for the randomly initialized weights with all metrics reporting a better uncertainty prediction. This is an anticipated result, as the random initialization allows the ensemble members to make more diverse predictions.

The major downside of the DE approach is the training time and the requirements on the hardware. If we only use small models for simple data sets, like the Noisy Sine in Chapter 4, this overhead is noticeable, but no further adaptations are necessary. For the training of Monodepth 2, however, the model is already considerably larger and newer transformer-based models require even more parameters. This in combination with a dataset containing high resolution images leads to very high computational cost. Additionally, the required gradients for the training occupy a lot of memory, making it difficult to train multiple models in parallel on a single GPU and limiting the size of the ensemble during inference.

5.4 REPULSIVE LAST LAYER ENSEMBLES (RLLES)

To address these challenges, we apply the RLLE [42] concept to Monodepth 2. The fact that only a part of the model is run as an ensemble reduces the overall computational cost. This is especially useful for large models, as the number of parameters only increases slightly compared to replicating the whole model multiple times. Another advantage is that the RLLE can be applied after the main model was already trained, and only re-training of the ensemble part is needed.

As the main model we use the architecture of Monodepth 2, but adjustments are necessary to allow the training of the RLLE. First a location inside the main network has to be chosen, where we can extract the intermediate feature values to input them into the ensemble. We have chosen the features before the last conventional layer of the last encoder block. In theory, every layer from the network could be used, but the resulting ensemble would have to be adjusted in complexity, as extracting the features earlier results in missing components like the residual connections and upsampling layers that need to be replaced.

The extracted features from the decoder have a dimension of $192 \times 640 \times 8$ and the target output has a dimension of $192 \times 640 \times 1$. Up until now we only used fully connected layers for the ensemble, but in this case one fully connected layer would result in around 120 billion parameters. This is not feasible, as the model would not fit into the memory of a single GPU especially during training, where the gradients also require memory. As a solution we extended the RLLE to also support FCNs, and higher dimensional outputs.

The ensemble architecture takes the 8 input channels, which are the feature output of the decoder, and applies one hidden layer with a

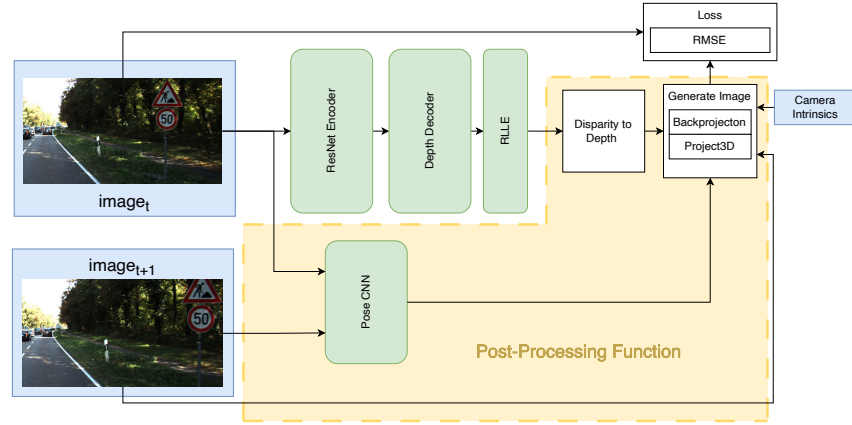


Figure 5.8: Training process of Monodepth 2 with a RLLE ensemble. The input image is passed through the encoder, the decoder and then the RLLE. The output disparity is then converted to depth, and combined with the estimated pose, to calculate the loss.

3×3 convolutions and 16 channels. After that the last convolutional layer is applied, which reduces the number of channels down to 1.

As described in Chapter 5 the training process of Monodepth 2 is complex due to the self-supervised learning approach. To enable the training of RLLE, some central adjustments are necessary that are shown in Figure 5.8. To calculate a loss, we need a generated image based on the predicted depth map and the pose change. We get a predicted disparity map as the output of our ensemble, which then can be converted to a depth map using the same method as before. To generate the images, we also need the pose estimation from the separate PoseNet. As this network is not part of the ensemble, we do not update the weights of the PoseNet and instead use the already trained weights. Given those two parts, we can then generate the images and calculate the loss. To allow the use of the RLLE ensemble, these steps, including the PoseNet, are combined into a complex post-processing function.

As the RLLE is currently not compatible with the multiple loss functions of Monodepth 2, we rely on the RMSE loss function. As the majority of the network is already trained, this should be sufficient. The training of the RLLE is done using the Adam optimizer with a learning rate of 0.001 for 5 epochs, a batch size of 12 and a repulsion strength γ of 1.

In contrast to the implementation of MCDO or DEs, RLLEs require significantly more adjustments to the existing code, as demonstrated here. With MCDO, only the dropout needs to be added, and with DEs, the same training can simply be repeated several times. The self-supervised learning approach of Monodepth 2 in particular requires a precise understanding of the underlying processes and code adaptations in numerous locations.

Act	Abs Rel ↓	RMSE ↓	$\delta < 1.25$ ↑
ELU	0.1154	4.8826	0.8692
RELU	0.1171	4.9285	0.8629
SELU	0.1154	4.8643	0.8689
Sigmoid	0.1162	4.9136	0.8670
SiLU	0.1162	4.9151	0.8669
Sine	0.1153	4.8887	0.8684
Tanh	0.1158	4.8851	0.8680

Table 5.9: Depth Estimation Performance of RLLE with different activation functions on the KITTI dataset. The Sine performs optimally for the Abs Rel metric, while SELU is most successful for the RMSE and accuracy.

Act	Abs Rel		RMSE		$\delta \geq 1.25$	
	AUSE ↓	AURG ↑	AUSE ↓	AURG ↑	AUSE ↓	AURG ↑
ELU	0.0621	0.0122	3.3376	0.8087	0.0998	0.0195
RELU	0.0656	0.0092	4.1079	0.0655	0.1120	0.0126
SELU	0.0595	0.0147	3.2036	0.9234	0.0913	0.0282
Sigmoid	0.0615	0.0132	3.7056	0.4642	0.0974	0.0238
SiLU	0.0548	0.0198	2.9066	1.2659	0.0819	0.0393
Sine	0.0463	0.0277	2.2835	1.8676	0.0637	0.0563
Tanh	0.0588	0.0155	3.3020	0.8431	0.0914	0.0289

Table 5.10: Uncertainty Prediction Performance of RLLE with different activation functions on the KITTI dataset: The Sine activation function performs best compared to all other activation functions across all metrics.

5.4.1 Activation Functions

Since we are now adding a new ensemble part and have already seen that the choice of activation function can have a big impact on RLLEs, we will first try to determine which activation function produces the best results.

The results of the depth prediction performance can be seen in Table 5.9 with all activation functions performing similar to the baseline. There is no overall best activation function, in terms of the depth prediction performance, instead it depends on the metric used. Adding performance of the uncertainty prediction, that is shown in Table 5.10, we can see that here the Sine activation function performs best for all shown metrics. The other two activation functions perform similar to each other, except for RELU with an especially bad performance for all

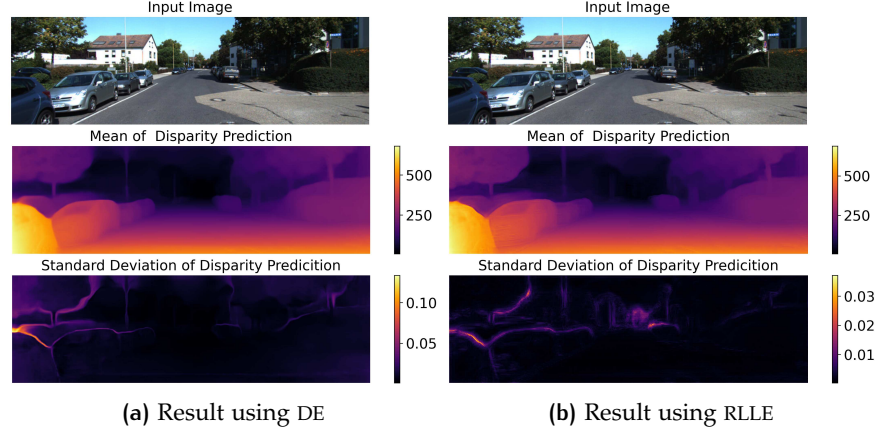


Figure 5.9: Comparison of the depth prediction performance of DE and RLLE: The two approaches yield similar results in terms of depth estimation, however, their uncertainty predictions differ, particularly for objects far away.

AURG metrics. As the Sine function performs well in the depth prediction and is the best performing activation function for the uncertainty prediction, we will use it for the next experiments.

When we visually compare the depth prediction of the RLLE with the DE in Figure 5.9 we can see that the depth prediction of the RLLE is similar, with the notable difference, that the RLLE produces some artifacts in the depth map. This is most likely because the RLLE is not trained with the smoothness loss function used in the regular Monodepth 2 training process. This missing smoothness also can be seen in the uncertainty prediction, where the RLLE tends to predict uncertainty maps with more focus on sharp edges. To what extent this is worse, or better behavior is not intuitively clear.

5.4.2 Repulsion Strength

To improve the uncertainty prediction performance further we can also vary the scaling of the repulsive force using γ value. As there is no theoretical background for the choice of the repulsion strength, we can only rely on empirical results. The results of the depth prediction performance can be seen in Table 5.11 and the uncertainty prediction performance in Table 5.12.

The results of the depth prediction performance reveal that the performance stays almost constant for the different γ values without a clear trend increasing or decreasing. The low variance of the depth prediction performance indicates that the performance is not sensitive to the choice of the repulsion strength.

Varying the repulsion strength does not provide a completely unambiguous trend for the uncertainty prediction performance, however,

γ	Abs Rel $\downarrow$	RMSE $\downarrow$	$\delta < 1.25$ $\uparrow$
$1e-5$	<u>0.1148</u>	4.8889	0.8694
$1e-4$	0.1157	4.9550	0.8652
$1e-3$	0.1153	4.8925	<u>0.8705</u>
$1e-2$	0.1151	4.9168	0.8676
$1e-1$	0.1164	4.9300	0.8675
$1e0$	0.1153	4.8887	0.8684
$1e1$	0.1150	<u>4.8450</u>	0.8702
$1e2$	0.1151	4.8785	0.8692
$1e3$	0.1158	4.9204	0.8662
$1e4$	0.1172	4.8562	0.8654

Table 5.11: Depth Estimation Performance of RLLE with different γ values on the KITTI dataset: Depending on the metric, there are varying optimal values for γ , although no clear trend is evident.

γ	Abs Rel		RMSE		$\delta \geq 1.25$	
	AUSE $\downarrow$	AURG $\uparrow$	AUSE $\downarrow$	AURG $\uparrow$	AUSE $\downarrow$	AURG $\uparrow$
$1e-5$	0.0539	0.0200	2.7630	1.3922	0.0807	0.0384
$1e-4$	0.0487	0.0253	2.2399	1.9683	0.0665	0.0562
$1e-3$	0.0540	0.0205	2.7788	1.3823	0.0802	0.0379
$1e-2$	<u>0.0402</u>	<u>0.0340</u>	<u>1.7306</u>	<u>2.4489</u>	<u>0.0495</u>	<u>0.0712</u>
$1e-1$	0.0475	0.0270	2.1490	2.0340	0.0653	0.0555
$1e0$	0.0463	0.0277	2.2835	1.8676	0.0637	0.0563
$1e1$	0.0528	0.0213	2.5064	1.6066	0.0745	0.0438
$1e2$	0.0523	0.0216	2.6974	1.4458	0.0769	0.0424
$1e3$	0.0497	0.0243	2.6516	1.5230	0.0715	0.0503
$1e4$	0.0505	0.0245	2.8289	1.2783	0.0705	0.0518

Table 5.12: Uncertainty Prediction Performance of RLLE with different γ values on the KITTI dataset: For $\gamma = 0.01$, all uncertainty prediction metrics are optimal, particular for the AURGs.

Size	Abs Rel ↓	RMSE ↓	$\delta < 1.25$ ↑
2	0.1158	4.9589	0.8664
4	0.1155	4.8612	0.8713
6	0.1150	4.8916	0.8696
8	0.1160	5.0060	0.8636
10	0.1151	4.9013	0.8683
20	0.1153	4.9448	0.8680

Table 5.13: Depth Estimation Performance of RLLE with different ensemble sizes on the KITTI dataset: The ensemble size seems to have no influence on the depth estimation

Size	Abs Rel		RMSE		$\delta \geq 1.25$	
	AUSE ↓	AURG ↑	AUSE ↓	AURG ↑	AUSE ↓	AURG ↑
2	0.0607	0.0138	3.3521	0.8631	0.0979	0.0238
4	0.0482	0.0266	2.3961	1.7358	0.0648	0.0526
6	0.0528	0.0213	2.4829	1.6747	0.0758	0.0432
8	0.0494	0.0250	2.5230	1.7314	0.0721	0.0520
10	0.0403	0.0339	1.7314	2.4346	0.0495	0.0706
20	0.0510	0.0233	2.5246	1.6817	0.0749	0.0455

Table 5.14: Uncertainty Prediction Performance of RLLE with different ensemble sizes on the KITTI dataset: For the ensemble size of 10 used during the calibration, the uncertainty prediction is best.

it shows a tendency for the best performance to be achieved with a γ of 0.01. This is especially noticeable for the AUSE and AURG of the RMSE and accuracy metrics.

5.4.3 Ensemble Size

Analogous to DEs, RLLEs offer the option of adjusting the size of the ensemble as required. This is particularly interesting for large networks, as the computational cost is significantly higher. A larger ensemble results in more parameters and longer training and inference times.

The results of the depth prediction with different ensemble sizes in Table 5.13 do not show a clear optimum. While for DEs larger ensembles led to consistently better depth predictions, here the values are so close to each other that the differences can be explained by solely noise. This behavior could be due to the fact that all ensemble members repulse each other and thus the enlargement of the ensemble simultaneously drives the members apart.

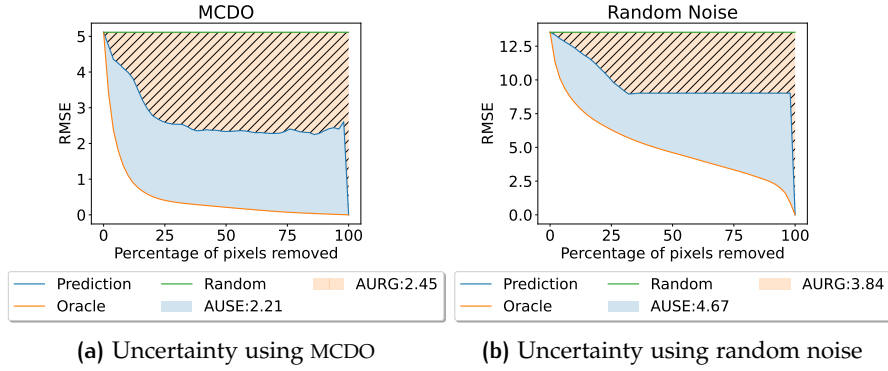


Figure 5.10: Example of the predicted uncertainty using MCDO (a) and random noise (b) for the RMSE-based AUSE and AURG. MCDO achieves a better AUSE score with 2.21, while random noise performs better for the AURG with 3.84.

With the uncertainty prediction in Table 5.14, however, there is a clear optimum choice for the ensemble size. With an ensemble size of $n = 10$ all metrics perform best. For larger and smaller ensembles, the uncertainty prediction performance decreases.

This is somewhat exceptional, as this behavior did not exist for the DEs either. One possible explanation is that this is an artifact of the optimization process we previously conducted with a given ensemble size of 10. This would mean that the choice of the other hyperparameters, especially the repulsive strength γ , is dependent on the ensemble size. On the one hand, this requires more effort to determine the best hyperparameters, on the other hand, it allows the search for optimal parameters for a given ensemble size. This can be useful for resource-constrained systems, and the ensemble is limited to a certain size.

5.5 CROSS-VALIDATION USING IMAGE CORRUPTION

As described in Section 5.1.3 the AUSE and AURG provide an indication of the predicted uncertainty in relation to the error of the depth prediction. This means that these metrics are not directly related to any comprehensible concept of uncertainty, except insofar as high uncertainty indicates when high prediction error is to be expected. For example, one might expect the model to predict strong uncertainty if it faces a previously unseen situation.

As an example, in Figure 5.10 we show two plots of the AUSE and AURG once for MCDO with a dropout rate of 0.1 and once where random noise is artificially added to the predicted depth map. We can

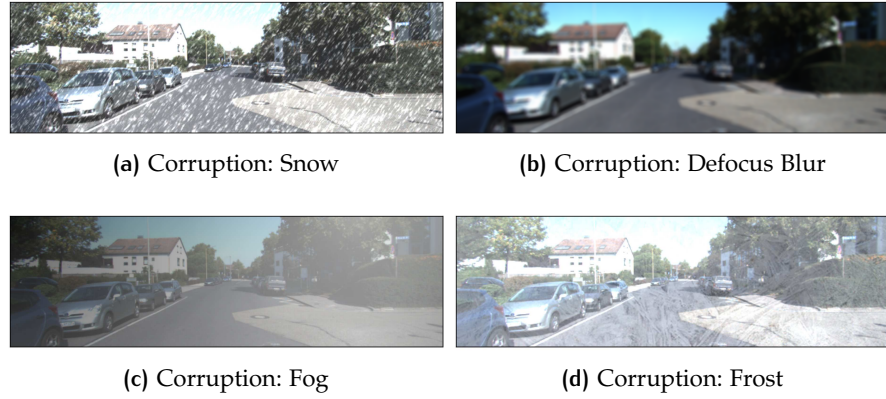


Figure 5.11: Visualization of different types of image corruption [19]: Snow, Defocus Blur, Fog and Frost.

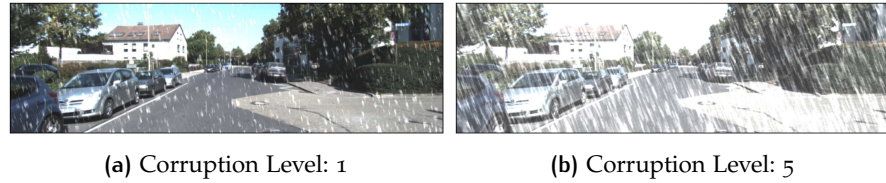


Figure 5.12: Visualization of two corruption levels using the snow corruption. A higher level of corruption changes the image significantly stronger.

see that the resulting AUSE is better for MCDO while the random noise outperforms it in terms of AURG.

By only analyzing the values of the metrics, one could intuitively interpret that the two approaches have a similarly good performance of uncertainty prediction. However, by analyzing the approaches logically, it is obvious that adding random noise cannot be a meaningful approach for uncertainty prediction. Therefore all results of the AUSE and AURG have to be taken with a grain of salt.

For this reason, we additionally evaluate the uncertainty prediction of the different models using artificially generated unseen images. To do this, we use image corruption [19] to generate realistic scenarios that could potentially occur but are not present in the training data. The 4 corruption types we have selected for this are shown in Figure 5.11.

In addition to choosing the type of corruption, we can also control the level of corruption applied. This ranges from 1 to 5, with 1 applying a low level of corruption and 5 applying the strongest level. Figure 5.12 shows an example image, once with low corruption and once with maximum corruption.

To allow a fair comparison between the methods, we do not compare individual uncertainty predictions but rather the relative changes in uncertainty as corruption increases. For each method, the mean

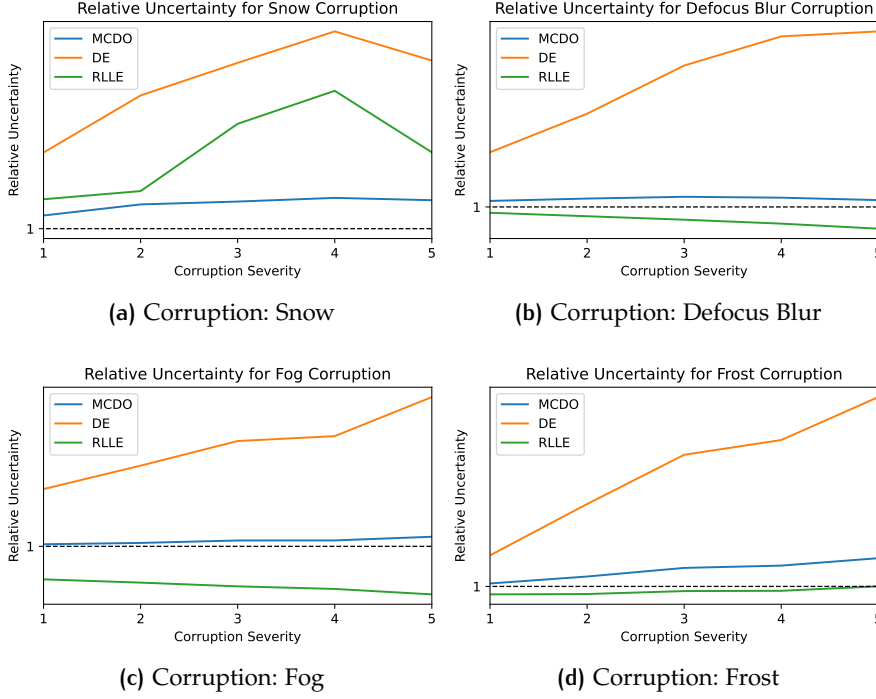


Figure 5.13: Relative uncertainty compared to uncorrupted images for different corruption types and methods: DEs show an increase in uncertainty for all corruption types, which is the desired behavior. MCDO only shows the desired behavior for snow (a) and frost (d) and RLLEs only for snow (a).

uncertainty at each corruption level is divided by the mean uncertainty of the uncorrupted image. This is averaged over the entire test data set. The expected outcome is that a higher corruption level results in an increase in relative uncertainty.

Figure 5.13 shows the resulting relative uncertainties for the four corruption types. The results reveal, that DEs report increased uncertainties the higher the corruption level is chosen, which is the desired behavior. Depending on the type of corruption, MCDO shows significantly smaller increase or remains almost at the same level of uncertainty for the defocus blur. Apart from the snow corruption, RLLEs fail at the task and are more certain in the case of corrupted images than in the case of uncorrupted ones. MCDO and RLLEs therefore seem to have difficulties in recognizing uncertainties in previously unseen situations.

5.6 HARDWARE PERFORMANCE

In addition to qualitative analyses of depth and uncertainty performance, it is also important to consider the hardware performance of individual concepts. The results of the parameter analysis and runtime

Method	# of Parameters	Training Time [s]	Inference Time per Image [ms]
MCDO	14,842,236	26,002	121.77
DE	133,580,124	244,173	41.42
RLLE	14,866,886	45,158	27.67

Table 5.15: Comparison of the hardware requirements and performance of the different approaches: DEs require significantly more parameters and training time, but MCDO takes significantly longer for inference. RLLEs are in between in terms of the number of parameters and training time and have the fastest inference time.

measurements of the different methods are shown in Table 5.15. To ensure comparability between results, all measurements were taken on a system with an AMD EPYC 7452 and an NVIDIA A30 GPU. The number of parameters is calculated based on the Monodepth 2 model, for DE for the multiple models and for RLLE based on the main model and the additional ensemble. The training time of the DE was measured serially. For the RLLE, the training time of the regular training process and the training time of the ensemble were added together. However, if there are already pre-trained weights for a model, this time would be reduced accordingly.

MCDO offers the advantage that the training process is only run once, which leads to the shortest training time, and that the number of parameters is the same as that of the base model. Sampling during inference, however, leads to a significantly increased time consumption, as the model must be run once for each sample. DEs require the most parameters and the longest training time, which could, however, be completely parallelized. The inference requires all models to be run, although these are usually fewer than for MCDO. RLLE has only marginally more parameters than the base model but requires additional training time for the ensemble. However, since the ensemble part is comparatively small, the inference time is the shortest.

5.7 CONCLUSION

Comparing the methods in terms of depth prediction performance in Table 5.16 shows that the DE with the pre-initialized weights of the ResNet performs best with the lowest errors. The randomly initialized weights for a DE perform significantly worse. The MCDO and RLLE approach performs slightly worse than the DE with pre-initialized weights, but better than the DE with randomly initialized weights.

Depending on the concrete metric MCDO performs slightly better than RLLE or vice versa. Somewhat surprisingly, MCDO with the added

Method	Abs Rel ↓	RMSE ↓	$\delta < 1.25$ ↑
Monodepth 2	0.1166	4.8560	0.8725
MCDO	0.1185	4.8549	0.8694
DE (Pretrained)	<u>0.1110</u>	<u>4.6340</u>	<u>0.8816</u>
DE (Scratch)	0.1269	4.9479	0.8535
RLLE	0.1151	4.9013	0.8683

Table 5.16: Comparison of Depth Estimation Performance between different methods on the KITTI dataset: The pre-trained DE provides the best depth estimation. The RLLE and MCDO perform slightly worse but about equally well.

Method	Abs Rel		RMSE		$\delta \geq 1.25$	
	AUSE ↓	AURG ↑	AUSE ↓	AURG ↑	AUSE ↓	AURG ↑
MCDO	0.0428	<u>0.0343</u>	2.1895	1.9223	0.0511	0.0676
DE (Pretrained)	0.0559	0.0160	3.6471	0.2863	0.0715	0.0371
DE (Scratch)	0.0538	0.0275	3.2714	0.8738	0.0656	0.0648
RLLE	<u>0.0403</u>	0.0339	<u>1.7314</u>	<u>2.4346</u>	<u>0.0495</u>	<u>0.0706</u>

Table 5.17: Comparison of Uncertainty Prediction Performance between different methods on the KITTI dataset: The uncertainty estimate of the RLLE is the best for almost all metrics.

noise and RLLE with the adjusted activation function and the modified loss function perform almost as well as the baseline model. This is a positive result, especially for RLLEs, because it shows that it is possible to apply this approach and obtain good results, even if not all concepts of the original training process can be adopted.

Comparing the uncertainty predictions in Table 5.17 reveal that contrary to the depth prediction performance, the DE with and without pre-initialized weights perform the worst. A possible explanation could be that due to the lack of noise, all ensemble members output too similar predictions regardless of the varying initialization. In contrast to that, MCDO receives noise from the dropout, which leads to more diverse outputs and RLLE obtain diversity by enforcing diverse predictions. This diversity leads to a better uncertainty prediction performance for both approaches with RLLE outperforming MCDO across most metrics.

However, these results based on the AUSE and AURG metrics contradict the corruption-based evaluation. The corruption-based analysis has shown that the DEs have consistently delivered the best uncertainty estimate, while they performed rather poorly in the AUSE and AURG score. Vice versa MCDO and RLLEs perform better in the uncertainty estimation based on the AUSE and AURG metrics, but they fail to deliver a good uncertainty estimate in the corruption-based evaluation. This suggests that the established metrics used by several existing works might not be fully suitable for the uncertainty evaluation and comparison of concepts.

To determine the most suitable concept for an application, the hardware requirements and corresponding runtime should also be considered. Depending on the application, for example, if the inference has to be fast, DEs or RLLEs should be used. However, in scenarios where the training needs to be short, MCDO is probably a good choice.

6

DISCUSSION AND OUTLOOK

In this work we have investigated the performance of different uncertainty estimation methods in the context of depth estimation. For this purpose, we selected MCDO, DEs and RLLEs, compared these methods using the Noisy Sine as a toy regression problem and then applied it to the more complex depth regression.

To gain an understanding of the individual methods, we first applied them to the synthetic Noisy Sine dataset. In contrast to the depth estimation task, the synthetic generation of data allows a defined expectation of the predicted aleatoric and epistemic uncertainty. We evaluated the performances of the methods using multiple metrics both in terms of the model accuracy and the uncertainty estimation. In particular, we conducted an ablation study and analyzed the performance of the different methods. To this end, we proposed the Epistemic Uncertainty Coefficient (EUC) as a new metric, to enable the metric-based evaluation of epistemic uncertainty in addition to the existing aleatoric uncertainty evaluation.

For MCDO, it has been shown that in addition to the selection of a suitable activation function and a matching variance regularizer, the dropout rate in particular has a major influence on the uncertainty estimation. However, MCDO struggled to correctly predict epistemic uncertainty and separate it from aleatoric uncertainty.

Overall, the following procedure has proven to be a sensible way for all methods to select appropriate hyperparameters: Start by selecting a suitable activation function and then adjusting the variance regularizer to calibrate the aleatoric error. Afterwards adjust model-specific hyperparameters to enable the model to recognize epistemic uncertainty.

DEs also require a suitable activation function and variance regularizer to produce good results. They approach the target distribution more and more as the ensemble size increases. With an appropriate choice, DEs delivers good results, but the diversity of the ensemble members is solely depending on the random initialization. RLLEs offer the most flexibility because, in addition to the hyperparameters already mentioned, the repulsion strength has a direct influence on diversity. Due to this flexibility, given the right calibration, RLLEs deliver the best results for the simple regression task even outperforming DEs. However, the RLLEs hyperparameters must also be calibrated appropriately with corresponding additional effort in order to obtain the optimum results.

In addition to the Noisy Sine, we also analyzed the performance for the much more demanding task of depth estimation. We used the existing architecture and training process of Monodepth 2 and evaluated the depth estimation and uncertainty prediction with established metrics. Analogous to before, we optimized the method-specific parameters for MCDO, DE and RLLE, although the activation function, for example, was already predefined by Monodepth 2. It was noticeable that the optimal values for the dropout rate or the repulsive strength are significantly different from before. These hyperparameters must therefore be individually optimized for each new data set. The flexibility, which is offered especially by RLLEs, therefore requires a large hyperparameter search, which requires corresponding computing resources.

Overall, the results are similar to before, with DEs providing a superior depth estimation through averaging multiple models. However, the uncertainty estimation of RLLEs again slightly outperforms DEs and MCDO. We also evaluated the concepts with artificially corrupted images, where only DEs consistently delivered good results. It is important to note that the results of the metric-based uncertainty estimation are not consistent with those obtained from the evaluation using corrupted images. A possible explanation for this could be that the optimization using the uncertainty metrics mainly considers aleatoric uncertainty, as it does not perform any evaluation for data that is not present in the training data.

Another important finding is that the implementation of RLLEs for more complex training processes is significantly more demanding than the other concepts. When comparing the model sizes and runtimes, it is noticeable that MCDO has the fastest training and requires the fewest parameters, but takes longer to infer. DEs, on the other hand, take a long time to train and have many parameters, but are faster during inference. RLLEs are between the other methods in terms of training time and parameters and offer the fastest inference time.

The choice of the optimal concept therefore cannot be answered in general terms and depends on different factors. If the BNN should run on an embedded system in real time, such as for autonomous driving, the advantages of RLLEs would make them a logical choice. However, if larger data analyses are to be carried out in a data centers, the high resource requirements of DEs are less critical and the parallelizability during training can be used.

There is a wide range of possible topics for future work. Until now, a fixed ensemble size has always been assumed for the RLLEs and a systematic analysis would possibly provide meaningful insights. In addition, a simplified loss function was used for the depth estimation and an extension to the loss function of Monodepth 2 would have the potential to improve the prediction results even further.

Probably the most exciting topic would be a more detailed analysis of the difference in performance between metric-based and corruption-based uncertainty prediction evaluation. Since the metric-based evaluation is conducted exclusively on data from the original data set, the assumption would be that the metrics primarily evaluate aleatoric uncertainty. In contrast, the corrupted images are not present in the training data, which should therefore be epistemic uncertainty. However, since the hyperparameters of the models were optimized using the established metrics, it does not necessarily mean that the results of the corruption evaluation are conclusive. For this purpose, the development of a new metric would be useful in order to objectively evaluate the behavior and to be able to calibrate the methods appropriately. Furthermore, it might also prove useful to implement the aleatoric head [24] analogous to Noisy Sine for the depth estimation for RLLEs as well as MCDO and DEs. This would allow the methods to distinguish between aleatoric and epistemic uncertainty.

In summary, this thesis not only advances the practical deployment of cost-efficient BNNs for depth estimation but also delivers a thorough investigation into the strengths and limitations of uncertainty quantification techniques based on MCDO, DEs, and RLLEs, thereby paving the way for more reliable and efficient neural networks in safety-critical applications.

ACRONYMS

ABS REL Absolute Relative Difference

AURG Area Under the Random Gain

AUSE Area Under the Sparsification Error

BNN Bayesian Neural Network

DE Deep Ensemble

EUC Epistemic Uncertainty Coefficient

FCN Fully Convolutional Networks

MCD0 Monte Carlo Dropout

MCMC Markov Chain Monte Carlo

MSE Mean Squared Error

NLL Negative Log-Likelihood

RDE Repulsive Deep Ensemble

RLLE Repulsive Last Layer Ensemble

RMSE Root Mean Squared Error

UPCONV-BLOCK Upscale-Convolutional-Block

BIBLIOGRAPHY

- [1] Moloud Abdar et al. "A review of uncertainty quantification in deep learning: Techniques, applications and challenges." In: *Information Fusion* 76 (2021), pp. 243–297. ISSN: 15662535. DOI: 10.1016/j.inffus.2021.05.008.
- [2] Ahmad O. Aseeri. "Uncertainty-Aware Deep Learning-Based Cardiac Arrhythmias Classification Model of Electrocardiogram Signals." In: *Computers* 10.6 (2021), p. 82. DOI: 10.3390/computers10060082.
- [3] Reiner Birkel, Diana Wofk, and Matthias Müller. *MiDaS v3.1 – A Model Zoo for Robust Monocular Relative Depth Estimation*. 2023. DOI: 10.48550/arXiv.2307.14460.
- [4] Dan Cireşan, Ueli Meier, and Juergen Schmidhuber. "Multi-column Deep Neural Networks for Image Classification." In: (2012). DOI: 10.48550/arXiv.1202.2745.
- [5] Francesco D’Angelo and Vincent Fortuin. *Repulsive Deep Ensembles are Bayesian*. URL: <http://arxiv.org/pdf/2106.11642>.
- [6] Juan G. Diaz Ochoa, Lukas Maier, and Orsolya Csiszar. "Bayesian logical neural networks for human-centered applications in medicine." In: *Frontiers in bioinformatics* 3 (2023), p. 1082941. DOI: 10.3389/fbinf.2023.1082941.
- [7] Xingshuai Dong, Matthew A. Garratt, Sreenatha G. Anavatti, and Hussein A. Abbass. *Towards Real-Time Monocular Depth Estimation for Robotics: A Survey*. 2021. DOI: 10.48550/arXiv.2111.08600.
- [8] David Eigen, Christian Puhrsch, and Rob Fergus. *Depth Map Prediction from a Single Image using a Multi-Scale Deep Network*. 2014. DOI: 10.48550/arXiv.1406.2283.
- [9] Wolfgang Fruehwirt et al. *Bayesian deep neural networks for low-cost neurophysiological markers of Alzheimer’s disease severity*. 2018. DOI: 10.48550/arXiv.1812.04994.
- [10] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. *Deep Ordinal Regression Network for Monocular Depth Estimation*. DOI: CVPR. URL: <http://arxiv.org/pdf/1806.02446v1>.
- [11] Yarın Gal and Zoubin Ghahramani. *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*. 2015. DOI: 10.48550/arXiv.1506.02142.

- [12] Jakob Gawlikowski et al. "A survey of uncertainty in deep neural networks." In: *Artificial Intelligence Review* 56.S1 (2023), pp. 1513–1589. ISSN: 0269-2821. DOI: 10.1007/s10462-023-10562-9.
- [13] A. Geiger, P. Lenz, and R. Urtasun. "Are we ready for autonomous driving? The KITTI vision benchmark suite." In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361. ISBN: 978-1-4673-1228-8. DOI: 10.1109/CVPR.2012.6248074.
- [14] Ethan Goan and Clinton Fookes. "Bayesian Neural Networks: An Introduction and Survey." In: *Case Studies in Applied Bayesian Data Science*. Ed. by Kerrie L. Mengersen, Pierre Pudlo, and Christian P. Robert. Vol. 2259. Lecture Notes in Mathematics. Cham: Springer International Publishing, 2020, pp. 45–87. ISBN: 978-3-030-42552-4. DOI: 10.1007/978-3-030-42553-1\textunderscore3.
- [15] Clément Godard, Oisín Mac Aodha, and Gabriel J. Brostow. *Unsupervised Monocular Depth Estimation with Left-Right Consistency*. URL: <http://arxiv.org/pdf/1609.03677>.
- [16] Clément Godard, Oisín Mac Aodha, Michael Firman, and Gabriel Brostow. *Digging Into Self-Supervised Monocular Depth Estimation*. URL: <https://arxiv.org/pdf/1806.01260>.
- [17] W. K. Hastings. "Monte Carlo sampling methods using Markov chains and their applications." In: *Biometrika* 57.1 (1970), pp. 97–109. ISSN: 0006-3444. DOI: 10.1093/biomet/57.1.97.
- [18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/arXiv.1512.03385.
- [19] Dan Hendrycks and Thomas G. Dietterich. *Benchmarking Neural Network Robustness to Common Corruptions and Surface Variations*. 2018. DOI: 10.48550/arXiv.1807.01697.
- [20] Julia Hornauer and Vasileios Belagiannis. "Gradient-based Uncertainty for Monocular Depth Estimation." In: 13680 (2022), pp. 613–630. DOI: 10.1007/978-3-031-20044-1\textunderscore35. URL: <http://arxiv.org/pdf/2208.02005v1>.
- [21] Eddy Ilg, Özgün Çiçek, Silvio Galesso, Aaron Klein, Osama Makansi, Frank Hutter, and Thomas Brox. *Uncertainty Estimates and Multi-Hypotheses Networks for Optical Flow*. URL: <http://arxiv.org/pdf/1802.07095>.
- [22] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. *Spatial Transformer Networks*. URL: <http://arxiv.org/pdf/1506.02025v3>.

- [23] Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. "Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users." In: *IEEE Computational Intelligence Magazine* 17.2 (2022), pp. 29–48. ISSN: 1556-603X. DOI: 10.1109/MCI.2022.3155327. URL: <https://arxiv.org/pdf/2007.06823>.
- [24] Alex Kendall and Yarin Gal. *What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?* 2017. URL: <http://arxiv.org/pdf/1703.04977>.
- [25] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. *Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles*. 2016. URL: <http://arxiv.org/pdf/1612.01474>.
- [26] E. L. Lehmann and Juliet Popper Shaffer. "Inverted Distributions." In: *The American Statistician* 42.3 (1988), p. 191. ISSN: 00031305. DOI: 10.2307/2684999.
- [27] Yucen Lily Li, Tim G. J. Rudner, and Andrew Gordon Wilson. *A Study of Bayesian Neural Network Surrogates for Bayesian Optimization*. 2023. DOI: 10.48550/arXiv.2305.20028.
- [28] Chao Liu, Jinwei Gu, Kihwan Kim, Srinivasa G. Narasimhan, and Jan Kautz. "Neural RGB@D Sensing: Depth and Uncertainty From a Video Camera." In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 10978–10987. ISBN: 978-1-7281-3293-8. DOI: 10.1109/CVPR.2019.01124.
- [29] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. "A survey of deep neural network architectures and their applications." In: *Neurocomputing* 234 (2017), pp. 11–26. ISSN: 09252312. DOI: 10.1016/j.neucom.2016.12.038.
- [30] Jonathan Long, Evan Shelhamer, and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2014. DOI: 10.48550/arXiv.1411.4038.
- [31] Rémi Marsal, Florian Chabot, Angelique Loesch, William Grolleau, and Hichem Sahbi. *MonoProb: Self-Supervised Monocular Depth Estimation with Interpretable Uncertainty*. 2023. DOI: 10.48550/arXiv.2311.06137.
- [32] Moritz Menze and Andreas Geiger. "Object scene flow for autonomous vehicles." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015, pp. 3061–3070. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298925.

- [33] D. A. Nix and A. S. Weigend. "Estimating the mean and variance of the target probability distribution." In: *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*. IEEE, 6/28/1994 - 7/2/1994, 55–60 vol.1. ISBN: 0-7803-1901-X. DOI: 10.1109/ICNN.1994.374138.
- [34] Andrew J. Parker and Ifan Betina Ip. "Stereoscopic Vision." In: *Reference Module in Neuroscience and Biobehavioral Psychology*. Elsevier, 2017. ISBN: 9780128093245. DOI: 10.1016/B978-0-12-809324-5.02958-8.
- [35] Emanuel Parzen. "On Estimation of a Probability Density Function and Mode." In: *The Annals of Mathematical Statistics* 33.3 (1962), pp. 1065–1076. ISSN: 0003-4851. DOI: 10.1214/aoms/1177704472.
- [36] Matteo Poggi, Filippo Aleotti, Fabio Tosi, and Stefano Mattoccia. *On the uncertainty of self-supervised monocular depth estimation*. URL: <http://arxiv.org/pdf/2005.06209>.
- [37] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. DOI: 10.48550/arXiv.1505.04597.
- [38] Olga Russakovsky et al. *ImageNet Large Scale Visual Recognition Challenge*. 2014. DOI: 10.48550/arXiv.1409.0575.
- [39] Karthik Abinav Sankararaman, Sinong Wang, and Han Fang. *BayesFormer: Transformer with Uncertainty Estimation*. 2022. DOI: 10.48550/arXiv.2206.00826.
- [40] B. Schölkopf, S. Mika, C. C. Burges, P. Knirsch, K. R. Müller, G. Rätsch, and A. J. Smola. "Input space versus feature space in kernel-based methods." In: *IEEE transactions on neural networks* 10.5 (1999), pp. 1000–1017. ISSN: 1045-9227. DOI: 10.1109/72.788641.
- [41] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html>.
- [42] Sophie Steger, Christian Knoll, Bernhard Klein, Holger Fröning, and Franz Pernkopf. *Function Space Diversity for Uncertainty Prediction via Repulsive Last-Layer Ensembles*. URL: <https://arxiv.org/pdf/2412.15758>.
- [43] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey." In: *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329. ISSN: 0018-9219. DOI: 10.1109/JPROC.2017.2761740.
- [44] Andrew Gordon Wilson. *Deep Ensembles as Approximate Bayesian Inference*. 2019. URL: <https://cims.nyu.edu/~andrewgw/deeensembles/>.

- [45] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. *Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data*. 2024. DOI: 10.48550/arXiv.2401.10891.
- [46] Jiayu Yao, Weiwei Pan, Soumya Ghosh, and Finale Doshi-Velez. *Quality of Uncertainty Quantification for Bayesian Neural Network Inference*. 2019. DOI: 10.48550/arXiv.1906.09686.
- [47] Shingo Yashima, Teppei Suzuki, Kohta Ishikawa, Ikuro Sato, and Rei Kawakami. *Feature Space Particle Inference for Neural Network Ensembles*. 2022. DOI: 10.48550/arXiv.2206.00944.