A General Framework for Uncertainty Estimation in Deep Learning

Antonio Loquercio*[1], Mattia Segu*[1], and Davide Scaramuzza[1]

***Abstract*—Neural networks predictions are unreliable when the input sample is out of the training distribution or corrupted by noise. Being able to detect such failures automatically is fundamental to integrate deep learning algorithms into robotics. Current approaches for uncertainty estimation of neural networks require changes to the network and optimization process, typically ignore prior knowledge about the data, and tend to make over-simplifying assumptions which underestimate uncertainty. To address these limitations, we propose a novel framework for uncertainty estimation. Based on Bayesian belief networks and Monte-Carlo sampling, our framework not only fully models the different sources of prediction uncertainty, but also incorporates prior data information, e.g. sensor noise. We show theoretically that this gives us the ability to capture uncertainty better than existing methods. In addition, our framework has several desirable properties: (i) it is *agnostic* to the network architecture and task; (ii) it does not require changes in the optimization process; (iii) it can be applied to *already trained* architectures. We thoroughly validate the proposed framework through extensive experiments on both computer vision and control tasks, where we outperform previous methods by up to 23% in accuracy.**

***Index Terms*—Deep Learning in Robotics and Automation, Probability and Statistical Methods, AI-Based Methods.**

Supplementary Material

The video available at https://youtu.be/X7n-bRS5vSM shows qualitative results of our experiments. The project's code is available at: https://tinyurl.com/v3jb64k

I. Introduction

ROBOTS act in an uncertain world. In order to plan and make decisions, autonomous systems can only rely on noisy perceptions and approximated models. Wrong decisions not only result in the failure of the mission but might even put human lives at risk, e.g., if the robot is an autonomous car (Fig. I). Under these conditions, deep learning algorithms can be fully integrated into robotic systems only if a measure of prediction uncertainty is available. Indeed, estimating uncertainties enables Bayesian sensor fusion and provides valuable information during decision making [1].

Manuscript received: September, 10th, 2019; Revised November, 4th, 2019; Accepted January, 14th, 2020.

This paper was recommended for publication by Editor Tamim Asfour upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by the Swiss National Center of Competence Research Robotics (NCCR), through the Swiss National Science Foundation, and the SNSF-ERC starting grant.

[1]All authors are with the Dep. of Informatics and Neuroinformatics of the University of Zurich and ETH Zurich, Switzerland.

Digital Object Identifier (DOI): see top of this page.

*These two authors contributed equally. Order is alphabetical.

Fig. 1. A neural network trained for steering angle prediction can be fully functional on a clean image (left) but generate unreliable predictions when processing a corrupted input (right). In this work, we propose a general framework to associate each network prediction with an uncertainty (illustrated above in red) that allows to detect such failure cases automatically.

Prediction uncertainty in deep neural networks generally derives from two sources: *data* uncertainty and *model* uncertainty. The former arises because of noise in the data, usually caused by the sensors' imperfections. The latter instead is generated from unbalances in the training data distribution. For example, a rare sample should have higher model uncertainty than a sample which appears more often in the training data. Both components of uncertainty play an important role in robotic applications. A sensor can indeed never be assumed to be noise free, and training datasets cannot be expected to cover all the possible edge-cases.

Traditional approaches for uncertainty estimation model the network activations and weights by parametric probability distributions. However, these approaches are particularly difficult to train [2] and are rarely used in robotic applications. Another family of approaches estimate uncertainties through sampling [3]. Since they do not explicitly model data uncertainty, these methods generate over-confident predictions [4]. In addition, methods based on sampling generally disregard any relationship between data and model uncertainty, which increases the risk of underestimating uncertainties. Indeed, an input sample with large noise should have larger model uncertainty than the same sample with lower noise.

In this paper, we propose a novel framework for uncertainty estimation of deep neural network predictions. By combining Bayesian belief networks [5], [6], [7] with Monte-Carlo sampling, our framework captures prediction uncertainties better than state-of-the-art methodologies. In order to do so, we propose two main innovations with respect to previous works: the use of prior information about the data, e.g., sensor noise, to compute data uncertainty, and the modelling of the relationship between data and model uncertainty. We demonstrate both theoretically and experimentally that these two innovations allow our framework to produce higher quality

uncertainty estimates than state-of-the-art methods. In addition, our framework has some desirable properties: (i) it is *agnostic* to the neural network architecture and task; (ii) it does not require any change in the optimization or learning process, and (iii) it can be applied to an *already trained* neural network. These properties make our approach an appealing solution to learning-based perception or control algorithms, enabling them to be better integrated into robotic systems.

To show the generality of our framework, we perform experiments on four challenging tasks: end-to-end steering angle prediction, obstacle future motion prediction, object recognition, and closed-loop control of a quadrotor. In these tasks, we outperform existing methodologies for uncertainty estimation by up to 23% in term of prediction accuracy. However, our framework is not limited to these problems and can be applied, without any change, to a wide range of tasks. Overall, our work makes the following contributions:

- We propose a general framework to compute uncertainties of neural networks predictions. Our framework is general in that it is agnostic to the network architecture, does not require changes in the learning or optimization process, and can be applied to already trained neural networks.
- We show mathematically that our framework can capture data and model uncertainty and use prior information about the data.
- We experimentally show that our approach outperforms existing methods for uncertainty estimation on a diverse set of tasks.

II. Related Work

In the following, we discuss the methods that have been proposed to estimate uncertainties and a series of approaches which have used this information in robotic systems.

A. Estimating Uncertainties in Neural Networks Predictions

A neural network is generally composed of a large number of parameters and non-linear activation functions, which makes the (multi-modal) posterior distribution of a network predictions intractable. To approximate the posterior, existing methods deploy different techniques, mainly based on Bayesian inference and Monte-Carlo sampling.

To recover probabilistic predictions, Bayesian approaches represent neural networks weights through parametric distributions, e.g., exponential-family [8], [5], [2], [9]. Consequently, networks' predictions can also be represented by the same distributions, and can be analytically computed using non-linear belief networks [5] or graphical models [10]. More recently, Wang et al. [9] propose natural parameter networks, which model inputs, parameters, nodes, and targets by Gaussian distributions. Overall, these family of approaches can recover uncertainties in a principled way. However, they generally increase the number of trainable parameters in a super-linear fashion, and require specific optimization techniques [2] which limits their impact in real-world applications.

In order to decrease the computational burden, Gast et al. [6] proposed to replace the network's input, activations, and outputs by distributions, while keeping network's weights deterministic. Similarly, probabilistic deep state space models retrieve data uncertainty in sequential data, and use it for learning-based filtering [11], [12]. However, disregarding weights uncertainty generally results in over-confident predictions, in particular for inputs not well represented in the training data.

Instead of representing neural networks parameters and activations by probability distributions, a second class of methods proposed to use Monte-Carlo (MC) sampling to estimate uncertainty. The MC samples are generally computed using an ensemble of neural networks. The prediction ensemble could either be generated by differently trained networks [13], [14], [15], or by keeping drop-out at test-time [3]. While this class of approaches can represent well the multi-modal posterior by sampling, it cannot generally represent data uncertainty, due for example to sensor noise. A possible solution is to tune the dropout rates [16], however it is always possible to construct examples where this approach would generate erroneous predictions [4].

To model data uncertainty, Kendall et al. [17] proposed to add to each output a "variance" variable, which is trained by a maximum-likelihood (a.k.a. heteroscedastic) loss on data. Combined with Monte-Carlo sampling, this approach can predict both the model and data uncertainty. However, this method requires to change the architecture, due to the variance addition, and to use the heteroscedastic loss for training, which is not always feasible.

Akin to many of the aforementioned methods, we use Monte-Carlo samples to predict model uncertainty. Through several experiments, we show why this type of uncertainty, generally ignored or loosely modelled by Bayesian methods [6], cannot be disregarded. In addition to Monte-Carlo sampling, our approach also computes the prediction uncertainty due to the sensors noise by using gaussian belief networks [5] and assumed density filtering [7]. Therefore, our approach can recover the full prediction uncertainty for any given (and possible already trained) neural network, without requiring any architectural or optimization change.

B. Uncertainty Estimation in Robotics

Given the paramount importance of safety, autonomous driving research has allocated a lot of attention to the problem of uncertainty estimation, from both the perception [18], [19] and the control side [13], [14]. Feng. et al. [18] showed an increase in performance and reliability of a 3D Lidar vehicle detection system by adding uncertainty estimates to the detection pipeline. Predicting uncertainty was also shown to be fundamental to cope with sensor failures in autonomous driving [13], and to speed-up the reinforcement learning process on a real robot [14].

For the task of autonomous drone racing, Kaufmann et al. [20] demonstrated the possibility to combine optimal control methods to a network-based perception system by using uncertainty estimation and filtering. Also for the task of robot manipulation, uncertainty estimation was shown to play a fundamental role to increase the learning efficiency and guarantee the manipulator safety [21], [22].

In order to fully integrate deep learning into robotics, learning systems should reliably estimate the uncertainty in their predictions [1]. Our framework represents a minimally invasive solution to this problem: we do not require any architectural changes or re-training of existing models.

III. Methodology

Due to the large number of (possibly non-linear) operations required to generate predictions, the posterior distribution $p(\mathbf{y}|\mathbf{x})$, where $\mathbf{y}$ are output predictions and $\mathbf{x}$ input samples, is intractable. Formally, we define the total prediction uncertainty as $\sigma_{tot} = \text{Var}_{p(\mathbf{y}|\mathbf{x})}(\mathbf{y})$. This uncertainty comes from two sources: data and model uncertainty. In order to estimate σ_{tot}, we derive a tractable approximation of $p(\mathbf{y}|\mathbf{x})$. In the following, we present the derivation of this approximation by using Bayesian inference, and the resulting algorithm to predict σ_{tot} (illustrated in Fig. 2).

A. The data uncertainty

Sensors' observations, e.g. images, are generally corrupted by noise. Therefore, what a neural network processes as input is $\mathbf{z}$, a noisy version of the "real" input $\mathbf{x}$. We assume that the sensor has known noise characteristic $\mathbf{v}$, which can be generally acquired by system identification or hardware specifications. Given $\mathbf{v}$, we assume the input data distribution $q(\mathbf{z}|\mathbf{x})$ to be:

$$q(\mathbf{z}|\mathbf{x}) \sim \mathcal{N}(\mathbf{z}; \mathbf{x}, \mathbf{v}) \,. \tag{1}$$

The output uncertainty resulting from this noise is generally defined as *data* (or aleatoric) uncertainty.

In order to compute data uncertainty, we forward-propagate sensor noise through the network via Assumed Density Filtering (ADF) [7]. This approach, initially applied to neural networks by Gast et al. [6], replaces each network activation, including input and output, by probability distributions. Specifically, the joint density of all activations in a network with l layers is:

$$p(\mathbf{z}^{(0:l)}) = p(\mathbf{z}^{(0)}) \prod_{i=1}^{l} p(\mathbf{z}^{(i)}|\mathbf{z}^{(i-1)}) \tag{2}$$

$$p(\mathbf{z}^{(i)}|\mathbf{z}^{(i-1)}) = \delta[\mathbf{z}^{(i)} - \mathbf{f}^{(i)}(\mathbf{z}^{(i-1)})] \tag{3}$$

where $\delta[\cdot]$ is the Dirac delta and $\mathbf{f}^{(i)}$ the i-th network layer. Since this distribution is intractable, ADF approximates it with:

$$p(\mathbf{z}^{(0:l)}) \approx q(\mathbf{z}^{(0:l)}) = q(\mathbf{z}^{(0)}) \prod_{i=1}^{l} q(\mathbf{z}^{(i)}) \tag{4}$$

where $q(\mathbf{z})$ is normally distributed, with all components independent:

$$q(\mathbf{z}^{(i)}) \sim \mathcal{N}\left(\mathbf{z}^{(i)}; \boldsymbol{\mu}^{(i)}, \mathbf{v}^{(i)}\right) = \prod_{j} \mathcal{N}\left(\mathbf{z}_j^{(i)}; \boldsymbol{\mu}_j^{(i)}, \mathbf{v}_j^{(i)}\right) . \tag{5}$$

The activation $\mathbf{z}^{(i-1)}$ is then processed by the (possibly non-linear) i-th layer function, $\mathbf{f}^{(i)}$, which transforms it into the (not necessarily normal) distribution:

$$\hat{p}(\mathbf{z}^{(0:i)}) = p(\mathbf{z}^{(i)}|\mathbf{z}^{(i-1)}) q(\mathbf{z}^{(0:i-1)}). \tag{6}$$

The goal of ADF is then to find the distribution $q(\mathbf{z}^{(0:i)})$ which better approximates $\hat{p}(\mathbf{z}^{(0:i)})$ under some measure, e.g. Kullback-Leibler divergence:

$$q(\mathbf{z}^{(0:i)}) = \underset{\hat{q}(\mathbf{z}^{(0:i)})}{\arg\min}\, \text{KL}(\hat{q}(\mathbf{z}^{(0:i)}) \parallel \hat{p}(\mathbf{z}^{(0:i)})) \tag{7}$$

Minka et al. [23] showed that the solution to (7) requires matching the moments of the two distributions. Under the normality assumptions, this is equivalent to:

$$\boldsymbol{\mu}^{(i)} = \mathbb{E}_{q(\mathbf{z}^{(i-1)})}[\, \mathbf{f}^{(i)}(z^{(i-1)})] \tag{8}$$

$$\mathbf{v}^{(i)} = \mathbb{V}_{q(\mathbf{z}^{(i-1)})}[\, \mathbf{f}^{(i)}(z^{(i-1)})] \tag{9}$$

where $\mathbb{E}$ and $\mathbb{V}$ are the first and second moment of the distribution. The solution of Eq. (8) and Eq. (9) can be computed analytically for the majority of functions used in neural networks, e.g. convolution, de-convolutions, relu, etc, and has an approximated solution for max-pooling. This results in a recursive formula to compute the activations mean and uncertainty, $(\boldsymbol{\mu}^{(i)}, \mathbf{v}^{(i)})$, given the parameters of the previous activations distribution $q(\mathbf{z}^{(i-1)})$. We refer the reader to [6], [5], [24] for the details of the propagation formulas.

In summary, ADF modifies the forward pass of a neural network to generate not only output predictions $\boldsymbol{\mu}^{(l)}$, but also their respective data uncertainties $\mathbf{v}^{(l)}$. In order to do so, ADF propagates the input uncertainty $\mathbf{v} = \mathbf{v}^{(0)}$, which, in a robotics scenario, corresponds to the sensor noise characteristics.

B. The model uncertainty

Model (or epistemic) uncertainty refers to the confidence a model has about its prediction. Similarly to Bayesian approaches [25], [26], [27], [28], [3], we represent this uncertainty by placing a distribution over the neural network weights, $\boldsymbol{\omega}$. This distribution depends on the training dataset $\text{D} = \{\mathbf{X}, \mathbf{Y}\}$, where $\mathbf{X}, \mathbf{Y}$ are training samples and labels, respectively. Therefore, the weight distribution after training can be written as $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$.

Except in trivial cases, $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$ is intractable. In order to approximate this distribution, Monte-Carlo based approaches collect weights samples by using dropout at test time [28], [3], [17]. Formally, this entails to approximate:

$$p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) \approx q(\boldsymbol{\omega}; \Phi) = Bern(\boldsymbol{\omega}; \Phi) \tag{10}$$

where Φ are the Bernoulli (or dropout) rates on the weights. Under this assumption, the model uncertainty is the variance of T Monte-Carlo samples, i.e. [3]:

$$\text{Var}^{model}_{p(\mathbf{y}|\mathbf{x})}(\mathbf{y}) = \sigma_{model} = \frac{1}{T} \sum_{t=1}^{T} (\mathbf{y}_t - \bar{\mathbf{y}})^2 \tag{11}$$

where $\{\mathbf{y}_t\}_{t=1}^{T}$ is a set of T sampled outputs for weights instances $\boldsymbol{\omega}^t \sim q(\boldsymbol{\omega}; \Phi)$ and $\bar{\mathbf{y}} = 1/T \sum_t \mathbf{y}_t$.

Eq. 11 has an intuitive explanation: Due to over-parametrization, a network develops redundant representations of samples frequently observed in the training data. Because of the redundancy, predictions for those samples will remain approximately constant when a part of the network is switched off with dropout. Consequently, these samples will receive

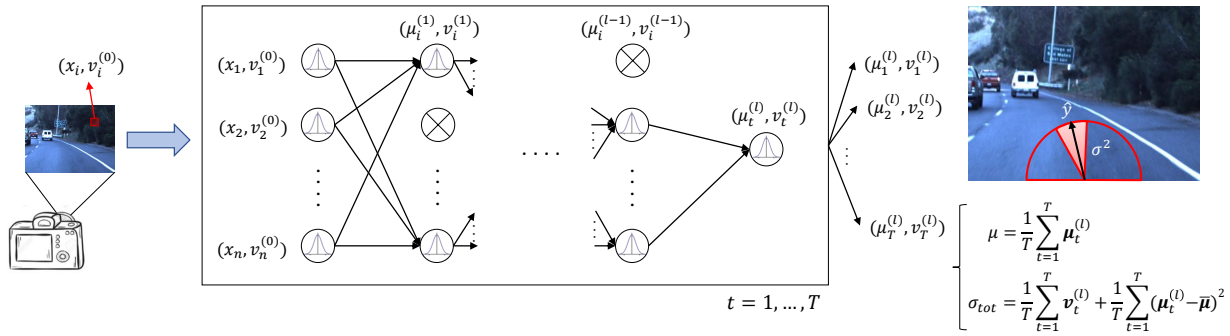

Fig. 2. Given an input sample $\mathbf{x}$, associated with noise $\mathbf{v}^{(0)}$, and a trained neural network, our framework computes the confidence associated to the network output. In order to do so, it first transforms the given network into a Bayesian belief network. Then, it uses an ensemble of T such networks, created by enabling dropout at test time, to generate the final prediction $\boldsymbol{\mu}$ and uncertainty σ_{tot}.

a low model uncertainty. In contrast, for rare samples the network is not able to generate such redundancies. Therefore, it will associate them with high model uncertainty.

C. Model uncertainty of an already trained network

The optimal dropout rates Φ needed to compute σ_{model} are the minimizers of the distance between the real and the hypothesis weight distribution:

$$\Phi = \arg\min_{\hat{\Phi}} \mathrm{KL}(p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) \parallel q(\boldsymbol{\omega}; \hat{\Phi})). \tag{12}$$

Previous works showed that the best Φ corresponds to the training dropout rates [28], [3], [17]. This, however, hinders the computation of model uncertainty for networks trained without dropout.

Since re-training a given network with a specific rate Φ is not always possible for several applications, we propose to find the best Φ *after training* by minimizing the negative log-likelihood between predicted and ground-truth labels. This is justified by the following lemma:

Lemma III.1. *The dropout rates Φ minimizing Eq. (12), under normality assumption on the output, are equivalent to:*

$$\Phi = \arg\min_{\hat{\Phi}} \sum_{d \in D} \frac{1}{2} \log(\sigma_{tot}^d) + \frac{1}{2\sigma_{tot}^d} (\mathbf{y}_{gt}^d - \mathbf{y}_{pred}^d(\hat{\Phi}))^2. \tag{13}$$

Proof. A successfully trained network $p_{net}(\mathbf{y}|\mathbf{x}, \boldsymbol{\omega})$ can very well predict the ground-truth, *i.e.*:

$$p_{gt}(\mathbf{y}|\mathbf{x}) \approx p_{pred}(\mathbf{y}|\mathbf{x}) = \int_{\boldsymbol{\omega}} p_{net}(\mathbf{y}|\mathbf{x}, \boldsymbol{\omega}) p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}). \tag{14}$$

By approximating $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$ by $q(\boldsymbol{\omega}|\Phi)$, *i.e.*, putting dropout only at test time, the real predicted distribution is actually:

$$\hat{p}_{pred}(\mathbf{y}|\mathbf{x}; \Phi) = \int_{\boldsymbol{\omega}} p_{net}(\mathbf{y}|\mathbf{x}, \boldsymbol{\omega}) q(\boldsymbol{\omega}|\Phi). \tag{15}$$

Since $p_{net}(\cdot)$ in Eq. (14) and Eq. (15) are the same, minimizing $\mathrm{KL}(p_{gt}(\mathbf{y}|\mathbf{x}) \parallel \hat{p}_{pred}(\mathbf{y}|\mathbf{x}; \Phi))$ is equivalent to minimizing Eq. (12). Assuming that both p_{net} and p_{gt} are normal, and that $\sigma_{gt} \to 0$ (i.e. ground-truth is quasi-deterministic), the distance between the predicted and ground-truth distribution is equivalent to Eq. (13). □

Practically, Φ is found by grid-search on a log-range of 20 possible rates in the range $[0, 1]$.

D. The total uncertainty

Section III-A shows that ADF can be used to propagate sensor uncertainties to the network outputs. This is equivalent to model the output distribution $p(\mathbf{y}|\mathbf{x}) \approx p(\mathbf{y}|\mathbf{z}, \boldsymbol{\omega}) p(\mathbf{z}|\mathbf{x})$, where $\boldsymbol{\omega}$ are deterministic network parameters and $p(\mathbf{z}|\mathbf{x})$ the sensor noise characteristics. Instead, Section III-B shows that model uncertainty can be computed by putting a distribution on the network weights $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$. The total uncertainty σ_{tot} results from the combination of the model and data uncertainty. It can be computed through a stochastic version of ADF, as presented in the following lemma.

Lemma III.2. *The total variance of a network output $\mathbf{y}$ for an input sample $\mathbf{x}$ corrupted by noise $\mathbf{v}^{(0)}$ is:*

$$\sigma_{tot} = Var_{p(\mathbf{y}|\mathbf{x})}(\mathbf{y}) = \frac{1}{T} \sum_{t=1}^{T} \mathbf{v}_t^{(l)} + (\boldsymbol{\mu}_t^{(l)} - \bar{\boldsymbol{\mu}})^2 \tag{16}$$

where $\{\boldsymbol{\mu}_t^{(l)}, \mathbf{v}_t^{(l)}\}_{t=1}^T$ is a set of T outputs from the ADF network with weights $\boldsymbol{\omega}^t \sim q(\boldsymbol{\omega}; \Phi)$ and $\bar{\boldsymbol{\mu}} = {}^1/_T \sum_t \boldsymbol{\mu}_t^l$.

Its proof can be found in the supplementary material. Intuitively, Eq. (16) generates the total uncertainty by summing the two components of data and model uncertainty. Note the difference between Eq. (11) and our model uncertainty, ${}^1/_T \sum_{t=1}^T (\boldsymbol{\mu}_t^{(l)} - \bar{\boldsymbol{\mu}})^2$. Differently from Eq. (11), the prediction ensemble used to calculate the model variance is not generated with network outputs $\mathbf{y}_t$, but with ADF predictions $\boldsymbol{\mu}_t^{(l)}$. Consequently, the model uncertainty also depends on the input sensor noise $\mathbf{v}^{(0)}$. Indeed, this is a very intuitive result: even though a sample has been frequently observed in the training data, it should have large model uncertainty if corrupted by high noise. From Lemma III.2 we derive a computationally feasible algorithm to compute, at the same time, predictions and total uncertainties. Illustrated in Fig. 2, the algorithm is composed of three main steps: (i) transforming a neural network into its ADF version (which does not require re-training), (ii) collect T samples by forwarding $(\mathbf{x}, \mathbf{v}^{(0)})$ to the network with $\boldsymbol{\omega}^t \sim q(\boldsymbol{\omega}; \Phi)$ and (iii) compute output predictions and variances according to lemma III.2.

It is interesting to draw a connection between Eq. (16) and the total uncertainty formulas used by previous works. Gast et al. [3], for example, do not use ADF networks to collect

Fig. 3. On well illuminated frames where the car is driving straight, a network trained to predict steering angles is very certain about its output (left). In contrast, for poorly illuminated, ambiguous frames, the network is highly uncertain about its predictions (right).

Monte-Carlo samples, and substitutes to the data uncertainty $\mathbf{v}^{(l)}$ a user-defined constant σ_d. Using a constant for the data uncertainty is nevertheless problematic, since different input samples might have different noise levels (due to, for example, temperature or pressure changes). Manually tuning this constant is generally difficult in practice, since it is not possible to use prior information about sensor noise characteristics. This makes it less attractive to robotics applications, where this information is either available or can be retrieved via identification.

In order to increase adaptation, Kendall et al. [17] proposed to learn the data uncertainty from the data itself. However, this comes at the cost of modifying the architecture and the training process, which hinders its application to already trained models and generally results in performance drops. Moreover, it considers the model and data uncertainty to be completely independent, which is a overly-restrictive assumption in many cases. For example, high sensor noise can result in large model uncertainty, in particular if the model was never exposed, at training time, to such kind of noise levels. In contrast, our approach can model this dependence, since it uses ADF samples to compute model uncertainty. We refer the reader to the proof of Lemma III.2 for the formal justification of the previous statements.

IV. Experiments

We validate our framework to compute uncertainties on several computer vision and robotic tasks. Specifically, as demonstrators we select end-to-end steering angle prediction, object future motion prediction, object recognition, and model-error compensation for autonomous drone flight. These demonstrators encompass the most active areas of research in mobile robotics, from computer vision to learning-based control. For each application, we compare against state-of-the-art methods for uncertainty estimation both qualitatively and quantitatively. All training details are reported in the appendix.

A. Demonstrators

End-to-End Steering Angle Prediction: Neural networks trained to predict steering angles from images have been used in several robotics applications, e.g. autonomous driving [29], [30] and flying [31]. In order to account for the safety of the platform, however, previous works showed the importance of quantifying uncertainties [14], [13]. In this section, we show that our framework can be used to estimate uncertainty without losing prediction performance on steering prediction.

Method	Re-train	RMSE	EVA	NLL
Gal et al. [3]	Yes	0.09	**0.83**	-0.72
Gast et al. [6]	Yes	0.10	0.79	-0.89
Kendall et al. [17]	Yes	0.11	0.75	**-1.1**
Ours	No	**0.09**	0.81	-1.0

TABLE I
BENCHMARK COMPARISON AGAINST STATE-OF-THE-ART METHODS FOR VARIANCE ESTIMATION ON THE END-TO-END STEERING ANGLE PREDICTION TASK.

To predict steering angles, we use the DroNet architecture of Loquercio et al. [31], since it was shown to allow closed-loop control on several unmanned aerial platforms [32], [33]. Differently from DroNet, however, we add a dropout layer after each convolution or fully connected layer. This is indeed necessary to extract Monte-Carlo samples. In order to show that our approach can estimate uncertainties from already trained networks, dropout is only activated *at test time*. We train this architecture on the Udacity dataset [31], which provides labelled images collected from a car in a large set of environments, illumination and traffic conditions. As it is the standard for the problem [29], [31], we train the network with mean-squared-error loss $\|y_{gt} - y_{pred}\|^2$, where y_{gt} and y_{pred} are the ground-truth and estimated steerings. For evaluation, we measure performance with Explained Variance (EVA) and Root Mean Squared Error (RMSE), as in [31]. Since there is no ground-truth for variances, we quantify their accuracy with negative log-likelihood (NLL) $1/2 \log(\sigma_{tot}) + 1/2\sigma_{tot}(y_{gt} - y_{pred})^2$ [3], [17], [6].

We compare our approach against state-of-the-art methods for uncertainty estimation. For a fair comparison, all methods share the same network architecture. For all the methods using sampling, we keep the number of samples fixed to $T = 20$, as it allows real-time performance (see Sec IV-B). Our approach additionally assumes an input noise variance $\mathbf{v} = 2$ grayscale levels, which is typical for the type of camera used in the Udacity dataset.

Table I summarizes the results of this experiment. Unsurprisingly, the method of Kendall et al. [17], trained to minimize the NLL loss, can predict good variances, but loses prediction quality due to the change of training loss and architecture. The approach of Gast et al. [6], trained under the same NLL objective, performs analogously in terms of RMSE and EVA. However, it performs worse in term of NLL, since this baseline only accounts for data uncertainty. In contrast to the previous baselines, the method of Gal et al. [3] predicts more precise steering angles due to ensembling, but generates poorer uncertainty estimates. With our framework, it is not necessary to make compromises: we can both make accurate predictions and have high quality uncertainty estimates, without changing or re-training the network.

Fig. 3 shows some qualitative results of our approach. As expected, our approach assigns very low variance to well-illuminated images with $y_{gt} \approx 0$. These are indeed the most frequent samples in the training dataset, and contain limited image noise. In contrast, our method predicts high uncertainties for images with large light gradients. This is

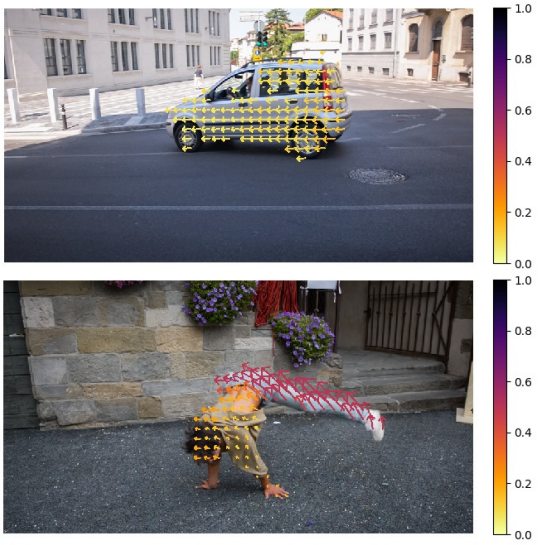

Fig. 4. Qualitative evaluation on the task of object future motion prediction. Future motion predictions are indicated by arrows, while uncertainties are color-coded. For objects whose motion is easily predictable, e.g car on the top, our framework produces good predictions with low uncertainties. In contrast, for object whose motion is not easily predictable, e.g. the dance at the bottom, predictions are associated with higher variance.

expected, since those samples can be ambiguous and have a smaller signal to noise ratio. For more qualitative experiments, we refer the reader to the supplementary video.

Object Future Motion Prediction: In this section, we train a neural network to predict the motion of an object in the future. Endowing robots with such an ability is important for several navigation tasks, e.g. path planning and obstacle avoidance. More specifically, the task is equivalent to predict the position of an object at time $t+\Delta t$, assuming to have the video of the object moving from $t = 0, \dots, t$. For the sake of simplicity, we predict motion only in the image plane, or, equivalently, the future 2D optical flow of the object.

In order to predict future flows we use the Flownet2S architecture [34], as it represents a good trade-off between performance and computational cost. The input to the network consists of a concatenation of the current and past frames I_t, I_{t-1}, and the object mask M_t. Its output is instead the optical flow *on the object* between the current and the (unobserved) future frame I_t, I_{t+1}. Ground-truth for this task is generated by the Flownet2 architecture [34], which is instead provided with the current and future frames I_t, I_{t+1}.

We perform experiments on the Davis 2016 dataset [35], which has high quality videos with moving objects, as well as pixel-level object masks annotations. Also for this experiment, it is assumed an input noise variance $\mathbf{v}^{(0)}$ of 2 pixels, which is compatible with the type of camera used to collect the dataset.

We again compare our framework for uncertainty estimation to state-of-the-art approaches [3], [6], [17]. To quantitatively evaluate the optical flow predictions, we use the standard end-point error (EPE) metric [34]. This metric is, however, 'local', since it does not evaluate the motion prediction *as a whole* but just as average over pixels. In order to better understand if our approach can predict the motion of the entire object correctly, we fit a Gaussian to both the sets of predicted and ground-truth flows. The KL distance between these two distributions represents our second metric. Finally, we use the standard negative log-likelihood metric (NLL) to evaluate the uncertainty estimations.

Method	Re-train	EPE	KL	NLL
Gal et al. [3]	Yes	5.99	56.7	6.96
Gast et al. [6]	Yes	6.12	50.1	5.74
Kendall et al. [17]	Yes	6.79	52.5	5.28
Ours	No	**5.91**	**45.1**	**4.07**

TABLE II
BENCHMARK COMPARISON AGAINST STATE-OF-THE-ART ON THE TASK OF OBJECT FUTURE MOTION PREDICTION.

Table II summarizes the results of this experiment. Our method outperforms all baselines on every metric. Interestingly, even though our network has not been specifically trained to predict variances as in Kendall et al. [17], it estimates uncertainty 23% better than the runner-up method. At the same time, being the network specifically trained for the task, it makes accurate predictions, outperforming the approach from Gal et al. [3] by 2% in terms of RMSE and 20% on the KL metric.

Qualitative results in Fig. 4 show that our framework captures an intuitive behaviour of the prediction network. Whenever the motion of the object is highly predictable, e.g. a car driving on the road, future optical flow vectors are accurately estimated and a low uncertainty is assigned to them. In contrast, if the object moves unpredictably, e.g. a dancer, the network is more uncertain about its predictions, particularly for the parts of the person which quickly change velocity.

Object Recognition: In this section, we investigate the performance of our framework on a classic computer vision task: object recognition. In order to do that, we evaluate our framework on the CIFAR-10 Dataset. We use two metrics to evaluate the performance of our approach: the average classification accuracy, and the average of per-class negative log-likelihood. Results of this evaluation are reported in Table III. Similarly to previous tasks, variance estimation in object recognition benefits from considering both model and data uncertainty. The aforementioned result table does not include the baseline of Kendall et al. [17], since its training procedure is specifically designed for regression problems, and it failed to converge in our classification experiments.

Closed-Loop Control of a Quadrotor: In this last experiment, we demonstrate that our framework can be used to fully integrate a deep learning algorithm into a robotics system. In order to do so, we consider the task of real-time, closed-loop control of a simulated quadrotor. For this task, we deploy a multi-layer perceptron (MLP) to learn compensation terms of a forward-dynamics quadrotor model. These compensations generally capture model inaccuracies, due to, e.g., rotor or fuselage drag [36].

We define the common model of a quadrotor, e.g. the one

Method	Re-train	Accuracy	NLL
Gal et al. [3]	Yes	93.2	4.79
Gast et al. [6]	Yes	93.7	15.2
Ours	No	**94.0**	**2.65**

TABLE III
BENCHMARK COMPARISON AGAINST STATE-OF-THE-ART ON THE TASK OF OBJECT RECOGNITION.

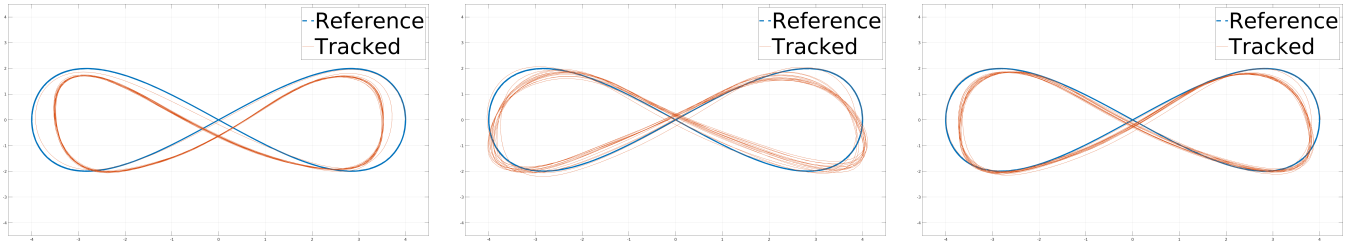

Fig. 5. Qualitative comparison between different control models in a lemniscate trajectory. Due to drag effects, the nominal model (left) cannot accurately track the reference trajectory. Linear model compensation [36] (middle) decreases tracking errors, but still provides suboptimal results. Our approach (right) providing non-linear compensations to the model only if the prediction uncertainty is within a confidence bound, achieves the best tracking performance.

of Mellinger et al. [37], as the *nominal* model. As proposed by previous work [36], we add to the linear and angular acceleration of the nominal model $\dot{p}, \dot{\omega}$ two compensations e_{lin}, e_{ang}. However, while previous work [36] predicts e_{lin}, e_{ang} as a linear function of the platform linear and angular speed v, ω, we propose to predict them as a function of the entire system state $s = (v, \dot{v}, \omega, \dot{\omega})$ via an MLP. Except for the modification of the function approximator (MLP instead of linear), we keep the training and testing methodology of Faessler et al. [36] unchanged.

We collect annotated training data to predict e_{lin}, e_{ang} in simulation [38]. Similarly to [36], we use a set of circular and lemniscate trajectories at different speeds to generate this data. The annotated data is then used to train our MLP, which takes as input s and outputs a 6 dimensional vector e_{lin}, e_{ang}.

We use our framework to compute the MLP's prediction uncertainty. At each time-step, if the uncertainty is larger than a user-defined threshold, the compensations will not be applied. This stops the network to compensate when uncertain, avoiding platform instabilities. Specifically, we set this threshold as five times the mean prediction uncertainty in an held-out testing set.

We perform closed-loop experiments on two types of trajectories: a horizontal circular with 4mt radius and an horizontal lemniscate (see Fig. 5). Both maneuvers, not observed at training time, were performed with a maximum speed of $7^{m}/_{s}$. Following previous work [36], we use the RMSE metric between the reference and actual trajectory for quantitative analysis. The results of this evaluation are presented in Table IV. Obviously, the high maneuvers' speed introduces significant drag effects, limiting the accuracy of the nominal model. Adding a linear compensation model, as in Faessler et al. [36], improves performance on the simple circular maneuver, but fails to generalize to the more involved lemniscate trajectory. Substituting the linear with a non-linear compensation model (MLP) improves generalization and boosts performance in the latter maneuver. However, applying the compensation only if the network is certain about its predictions (MLP with σ_{tot}) additionally increases tracking performance by 2% and 8% on the circular and lemniscate trajectories, respectively. Indeed, these maneuvers, unobserved at training time, contain states for which compensation is highly uncertain. Finally, Fig. 5 shows a qualitative comparison on the tracking performance of the different methods. Thanks to the non-linearities and the uncertainty estimation, our approach appears to be closer to the reference trajectory, hence minimizing tracking errors.

Method	Circular	Lemniscate
Nominal model	0.271	0.299
Faessler et al. [36]	0.086	0.298
MLP (Ours)	0.086	0.255
MLP with σ_{tot} (Ours)	**0.084**	**0.234**

TABLE IV
QUANTITATIVE COMPARISON ON TRAJECTORY TRACKING, CLOSED-LOOP EXPERIMENTS.

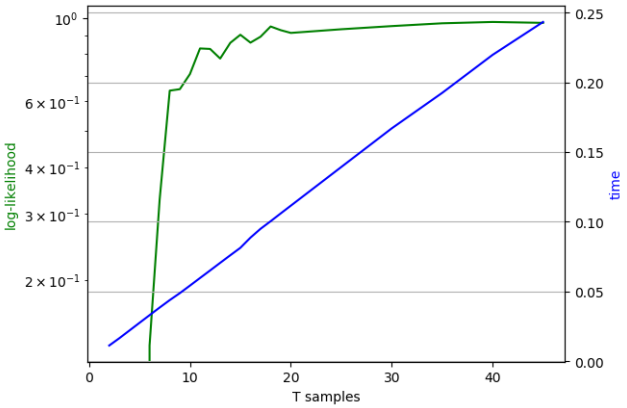

Fig. 6. As typical for sampling based approaches, a higher number of samples improves uncertainty estimates. In order to obtain real-time estimates, it is necessary to trade-off performance for speed. For example, in the task of end-to-end steering angle prediction, $T = 20$ samples is enough to have good uncertainty estimates and $\approx$ 10Hz inference rate.

B. Practical Considerations

Run-time Analysis: We perform a run-time analysis of our framework to study the trade-off between inference time and estimation accuracy. Similar to all methods based on Monte-Carlo sampling, our approach requires multiple forward passes of each image to estimate uncertainty. The larger the number of samples, the better the estimates [3]. As it can be observed in Fig. 6, the quality of variance estimation, measured in terms of NLL, plateaus for $T \geq N$ samples. In our experiments, we selected $T = 20$ as it allows processing at $\approx 10Hz$, which is acceptable for closed-loop control [32], [33].

Feed-forward vs Recurrent Models: Since our derivations are agnostic to the architecture, the proposed framework can be applied to both feed-forward and recurrent models. However, while recurrent models can improve performance on sequential tasks, their computational cost for extracting uncertainty is significantly higher: for each Monte-Carlo sample, the entire temporal sequence needs to be re-processed.

V. Conclusion

In this work, we present a general framework for uncertainty estimation of neural network predictions. Our framework is general in the sense that it is agnostic to the architecture, the learning procedure, and the training task. Inspired by Bayesian inference, we mathematically show that our approach tightly couples the sources of prediction uncertainty. To demonstrate the flexibility of our approach, we test it on several control and vision tasks. On each task we outperform state-of-the-art methods for uncertainty estimation, without compromising prediction accuracy.

Similarly to all sampling-based methods [3], [17], the main limitation of our approach is that, in order to generate σ_{tot}, we need several network forward passes for each input. This is particularly problematic for recurrent models, which need to unroll the entire temporal sequence for each sample. Although we show that this does not hinder real time performance (see Fig. 6), it still represents the main bottleneck of our framework. We believe that finding alternative solutions to compute model uncertainty, using, e.g., information theory [39], is a very interesting venue for future work.

References

[1] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and P. Corke, "The limits and potentials of deep learning for robotics," *Int. Journ. on Robotics Research*, vol. 37, no. 4-5, pp. 405–420, 2018.

[2] J. M. Hernández-Lobato and R. Adams, "Probabilistic backpropagation for scalable learning of bayesian neural networks," in *International Conference on Machine Learning*, 2015, pp. 1861–1869.

[3] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation:representing model uncertainty in deep learning," *Proceedings of the 33rd International Conference on Machine Learning*, 2015.

[4] O. Ian, "Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout," in *Advances in Neural Information Processing Systems Workshops*, 2016.

[5] Frey and Hinton, "Variational learning in nonlinear gaussian belief networks," *Neural Comput.*, vol. 11, no. 1, pp. 193–213, 1999.

[6] J. Gast and S. Roth, "Lightweight Probabilistic Deep Networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[7] K. Boyen, "Tractable inference for complex stochastic processes," *InUAI*, pp. 33–42, 1998.

[8] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," in *International Conference on Machine Learning*, 2015, pp. 1613–1622.

[9] H. Wang, S. Xingjian, and D.-Y. Yeung, "Natural-parameter networks: A class of probabilistic neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 118–126.

[10] Q. Su, X. Liao, C. Chen, and L. Carin, "Nonlinear statistical learning with truncated gaussian graphical models," in *International Conference on Machine Learning*, 2016, pp. 1948–1957.

[11] M. Fraccaro, S. Kamronn, U. Paquet, and O. Winther, "A disentangled recognition and nonlinear dynamics model for unsupervised learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 3601–3610.

[12] H. Wu, A. Mardt, L. Pasquali, and F. Noe, "Deep generative markov state models," in *Advances in Neural Information Processing Systems*, 2018, pp. 3975–3984.

[13] L. Keuntaek, W. Ziyi, V. B. I, B. Harleen, and T. Evangelos, "Ensemble bayesian decision making with redundant deep perceptual control policies," *ArXiv*, 2018.

[14] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[15] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Advances in Neural Information Processing Systems*, 2017, pp. 6402–6413.

[16] Y. Gal, J. Hron, and A. Kendall, "Concrete dropout," in *Advances in Neural Information Processing Systems*, 2017, pp. 3581–3590.

[17] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017.

[18] D. Feng, L. Rosenbaum, and K. Dietmayer, "Towards safe autonomous driving: Capture uncertainty in the deep neural network for lidar 3d vehicle detection," *Proceedings of the 21st IEEE International Conference on Intelligent Transportation Systems*, 2018.

[19] L. Neumann, A. Zisserman, and A. Vedaldi, "Relaxed softmax: Efficient confidence auto-calibration for safe pedestrian detection," in *Advances in Neural Information Processing Systems Workshops*, 2018.

[20] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Beauty and the beast: Optimal methods meet learning for drone racing," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[21] F. Stulp, E. Theodorou, J. Buchli, and S. Schaal, "Learning to grasp under uncertainty," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 5703–5708.

[22] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Advances in Neural Information Processing Systems*, 2018, pp. 4754–4765.

[23] T. P. Minka, "A family of algorithms for approximate bayesian inference," Ph.D. dissertation, Massachusetts Institute of Technology, 2001.

[24] J. D. Culurciello, "Robust convolutional neural networks under adversarial noise," *InWorkshop Proceedings of the ICLR*, 2016.

[25] J. Denker and Y. LeCun, "Transforming neural-net output levels to probability distributions," *Advances in Neural Information Processing Systems 3*, 1991.

[26] D. J. MacKay, "A practical Bayesian framework for backpropagation networks," *Neural Computation*, vol. 4, no. 3, pp. 448–472, 1992.

[27] R. M. Neal, "Bayesian learning for neural networks," *PhD thesis, University of Toronto*, 1995.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, 2014.

[29] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[30] M. Müller, A. Dosovitskiy, B. Ghanem, and V. Koltun, "Driving policy transfer via modularity and abstraction," in *Conference on Robot Learning (CoRL)*, 2018.

[31] A. Loquercio, A. I. Maqueda, C. R. D. Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, 2018.

[32] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "A 64mw dnn-based visual navigation engine for autonomous nano-drones," *IEEE Internet of Things Journal*, 2019.

[33] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: Learning agile flight in dynamic environments," in *Conference on Robot Learning (CoRL)*, 2018.

[34] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2462–2470.

[35] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. V. Gool, M. Gross, and A. Sorkine-Hornung, "A benchmark dataset and evaluation methodology for video object segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[36] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, 2018.

[37] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[38] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "RotorS–a modular gazebo MAV simulator framework," in *Robot Operating System (ROS)*. Springer, 2016, pp. 595–625.

[39] A. Achille and S. Soatto, "Where is the information in a deep neural network?" *arXiv preprint arXiv:1905.12213*, 2019.

[40] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *Proceedings of the IEEE*

Conference on Computer Vision and Pattern Recognition, 2016, pp. 4040–4048.

[41] W. Wenguan, S. Hongmei, Z. Shuyang, S. Jianbing, Z. Sanyuan, H. Steven, and L. Haibin, "Learning unsupervised video object segmentation through visual attention," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3064–3074.

VI. Supplementary Material

A. Proof of Lemma III.2

Consider the probabilistic model of the ADF network and the probabilistic distribution over the input:

$$\begin{aligned} p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega}) &= p(\mathbf{y}|\mathbf{z}, \boldsymbol{\omega}) p(\mathbf{z}|\mathbf{x}) \\ p(\mathbf{z}|\mathbf{x}) &\sim \mathcal{N}\left(\mathbf{z}; \mathbf{x}, \mathbf{v}_t^{(0)}\right) \end{aligned} \tag{17}$$

Let's now place a posterior distribution $p\left(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}\right)$ over network weights given the training data $\mathbf{D} = \{\mathbf{X}, \mathbf{Y}\}$. Consequently, the full posterior distribution of the Bayesian ADF network can be parametrized as

$$\begin{aligned} p\left(\mathbf{y}, \mathbf{z}|\mathbf{x}, \mathbf{X}, \mathbf{Y}\right) &= \left(\int p(\mathbf{y}|\mathbf{z}, \boldsymbol{\omega}) \cdot p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) d\boldsymbol{\omega} \right) \cdot p(\mathbf{z}|\mathbf{x}) \\ &= \int p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega}) \cdot p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) d\boldsymbol{\omega} \end{aligned} \tag{18}$$

where $p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega}) = p(\mathbf{y}|\mathbf{z}, \boldsymbol{\omega}) \cdot p(\mathbf{z}|\mathbf{x}) \sim \mathcal{N}(\widehat{\mathbf{y}}_{\boldsymbol{\omega}}, \mathbf{v}_t^{(l)} \mathbf{I}_D)$ for each model weights realization $\boldsymbol{\omega}$. Also, we approximate the intractable posterior over network weights as

$$p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) \approx q(\boldsymbol{\omega}) = \text{Bern}(\mathbf{z}_1) \cdots \text{Bern}(\mathbf{z}_L) \tag{19}$$

where $\text{Bern}(\mathbf{z}_i)$ is a Bernoullian distribution over the activation of the i-th layer. Thus,

$$p\left(\mathbf{y}, \mathbf{z}|\mathbf{x}, \mathbf{X}, \mathbf{Y}\right) \approx \int p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega}) \cdot q(\boldsymbol{\omega}) d\boldsymbol{\omega} = q(\mathbf{y}, \mathbf{z}|\mathbf{x}) \tag{20}$$

We will now prove that our framework actually recovers the total variance by plugging multiple stochastic forward passes with MC dropout in Lemma III.2.

Proof.

$$\begin{aligned} &\mathbb{E}_{q(\mathbf{y}, \mathbf{z}|\mathbf{x})}\left(\mathbf{y}\mathbf{y}^T\right) \\ &\overset{(1)}{=} \int \Bigg(\int \mathbf{y}\mathbf{y}^T \cdot p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega}) \mathrm{d}\mathbf{y} \Bigg) q(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} \\ &\overset{(2)}{=} \int \Bigg(\text{Cov}_{p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega})}(\mathbf{y}_{\boldsymbol{\omega}}) \\ &\qquad + \mathbb{E}_{p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega})}(\mathbf{y}_{\boldsymbol{\omega}}) \mathbb{E}_{p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega})}(\mathbf{y}_{\boldsymbol{\omega}})^T \Bigg) \cdot q(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} \\ &\overset{(3)}{=} \int \Bigg(\text{Cov}_{p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega})}(\mathbf{y}_{\boldsymbol{\omega}}) + \mathbb{E}_{p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega})}(\mathbf{y}_{\boldsymbol{\omega}}) \mathbb{E}_{p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega})}(\mathbf{y}_{\boldsymbol{\omega}})^T \Bigg) \\ &\qquad \cdot \text{Bern}(\mathbf{z}_1) \cdots \text{Bern}(\mathbf{z}_L) \mathrm{d}\mathbf{z}_1 \cdots \mathrm{d}\mathbf{z}_L \\ &\overset{(4)}{=} \int \Bigg(\mathbf{v}_t^{(l)} \mathbf{I}_D + \widehat{\mathbf{y}}(\mathbf{x}, \mathbf{z}_1, ..., \mathbf{z}_L) \widehat{\mathbf{y}}(\mathbf{x}, \mathbf{z}_1, ..., \mathbf{z}_L)^T \Bigg) \\ &\qquad \cdot \text{Bern}(\mathbf{z}_1) \cdots \text{Bern}(\mathbf{z}_L) \mathrm{d}\mathbf{z}_1 \cdots \mathrm{d}\mathbf{z}_L \\ &\overset{(5)}{\approx} \frac{1}{T} \sum_{t=1}^{T} \mathbf{v}_t^{(l)} + \frac{1}{T} \sum_{t=1}^{T} \widehat{\mathbf{y}}(\mathbf{x}, \widehat{\mathbf{z}}_{1,t}, ..., \widehat{\mathbf{z}}_{L,t}) \widehat{\mathbf{y}}(\mathbf{x}, \widehat{\mathbf{z}}_{1,t}, ..., \widehat{\mathbf{z}}_{L,t})^T \end{aligned}$$

(1) follows by the definition of expected value.
(2) follows by the definition of covariance:

$$\text{Cov}(\mathbf{y}) = \mathbb{E}(\mathbf{y}\mathbf{y}^T) - \mathbb{E}(\mathbf{y}) \, \mathbb{E}(\mathbf{y})^T$$

(3) follows from Equation 19.
(4) since $p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega}) = \mathcal{N}\big(\mathbf{y}; \widehat{\mathbf{y}}(\mathbf{x}, \mathbf{z}_1, ..., \mathbf{z}_L), \mathbf{v}_t^{(l)} \mathbf{I}_D\big)$.
(5) approximation by Monte Carlo integration.

Consequently, from the result just obtained and by the definition of variance, it can be easily shown that the total variance can be computed as:

$$\begin{aligned} \text{Var}_{q(\mathbf{y}|\mathbf{x})}(\mathbf{y}) &= \mathbb{E}_{q(\mathbf{y}, \mathbf{z}|\mathbf{x})}\left(\mathbf{y}\mathbf{y}^T\right) - \mathbb{E}_{q(\mathbf{y}, \mathbf{z}|\mathbf{x})}(\mathbf{y}) \, \mathbb{E}_{q(\mathbf{y}, \mathbf{z}|\mathbf{x})}(\mathbf{y})^T \\ &\approx \frac{1}{T} \sum_{t=1}^{T} \widehat{\mathbf{y}}(\mathbf{x}, \widehat{\mathbf{z}}_{1,t}, ..., \widehat{\mathbf{z}}_{L,t}) \widehat{\mathbf{y}}(\mathbf{x}, \widehat{\mathbf{z}}_{1,t}, ..., \widehat{\mathbf{z}}_{L,t})^T \\ &\quad - \mathbb{E}_{q(\mathbf{y}, \mathbf{z}|\mathbf{x})}(\mathbf{y}) \, \mathbb{E}_{q(\mathbf{y}, \mathbf{z}|\mathbf{x})}(\mathbf{y})^T + \frac{1}{T} \sum_{t=1}^{T} \mathbf{v}_t^{(l)} \\ &= \frac{1}{T} \sum_{t=1}^{T} \mathbf{v}_t^{(l)} + (\boldsymbol{\mu}_t^{(l)} - \bar{\boldsymbol{\mu}})^2 \end{aligned}$$

The total variance of a network output $\mathbf{y}$ for an input sample $\mathbf{x}$ corrupted by noise $\mathbf{v}^{(0)}$ is:

$$\sigma_{tot} = \text{Var}_{p(\mathbf{y}|\mathbf{x})}(\mathbf{y}) = \frac{1}{T} \sum_{t=1}^{T} \mathbf{v}_t^{(l)} + (\boldsymbol{\mu}_t^{(l)} - \bar{\boldsymbol{\mu}})^2 \tag{21}$$

which indeed amounts to the sum of the sample variance of T MC samples *(model uncertainty)* and the average of the corresponding *data variances* $\mathbf{v}_t^{(l)}$ returned by the ADF network. □

In conclusion, the final output of our framework is $[\mathbf{y}^*, \sigma_{tot}]$, where $\mathbf{y}^*$ is the mean of the mean predictions $\{\hat{\mathbf{y}}_t\}_{t=1}^T$ collected over T stochastic forward passes.

VII. Training Details

A. Implementation

Our framework for uncertainty estimation is implemented in Pytorch, and included in the supplementary material. It will be publicly released upon acceptance. For training and testing, we use a desktop computer equipped with an NVIDIA-RTX 2080.

B. End-to-End Steering Angle Prediction

The NN architecture used for the task *End-to-End Steering Angle Prediction* is a shallow ResNet that takes inspiration from the DroNet architecture by Loquercio et al. [31]. The network was trained on the Udacity dataset [31], containing approximately $70,000$ images captured from a car and distributed over 6 different experiment settings, 5 for training and 1 for testing. A validation set is held out from the data of the first experiment. For every experiment, time-stamped images are stored from 3 cameras (left, central, right) with the associated data: IMU, GPS, gear, brake, throttle, steering angles and speed. For our purpose, only images from the forward-looking camera and their associated steering angles

are used. The network is trained for 100 epochs with Adam and an initial learning rate of $1e-3$. The loss used for training is an L2-loss, which is also computed on the validation set at every epoch to select the best model. The total training time on our aforementioned hardware amounts to 6 hours .

C. Object Future Motion Prediction

For the task *Object Future Motion Prediction* we employ a FlowNet2S architecture to predict the future optical flow. Given the frames at time $t-1$ and t, it predicts the future optical flow between frame t and $t+1$. We use the publicly available weights pre-trained on FlyingChairs and FlyingThings3D [40] datasets for the task of optical flow regression to initialize our model, which is then specifically trained for our task for 2000 epochs (around 15 hours on our hardware) on the DAVIS 2016 dataset [35]. We used Adam with a learning rate of $1e-3$ and the loss used is the multi-scale L1 metrics. This multi-scale L1 loss is computed only on the moving object, identified by the segmentation mask. As input to the network, we pass the frames at time $t-1$ and t, stacked together with the segmentation mask corresponding to frame t. The optical flow between image t and $t+1$ is used as ground-truth. Since DAVIS dataset does not provide optical flow annotations, we use the state-of-the-art FlowNet2 [34] architecture to collect optical flow annotations for this dataset. Instead, the segmentation masks used as auxiliary input are already provided along with the DAVIS dataset. At test time, to prove the efficacy of the proposed method, the object mask M_t is generated by the state-of-the-art object detector AGS [41]. This indeed provides a good indicator of the network performance 'in the wild', where no ground-truth object mask is available.

D. Model Error Compensation

The Multi Layer Perceptron (MLP) used for model error compensation in the task *Closed-Loop Control of a Quadrotor* was trained for 100 epochs on a dataset consisting of data collected by a drone. We used Adam with $1e-4$ as learning rate, together with an L1 loss. Training took approximately 2 hours on our hardware. As input we used 24 features, each of them being collected at the same time step. These features are quaternion odometry, linear velocity odometry, angular velocity odometry, and thrusts. The network was trained to learn the linear and angular model error. The data were collected from three different kind of trajectories: circular, lemniscate and random. The circular and lemniscate trajectories were generated with a fixed radius of $4m$ and different velocities. Eventually, the training dataset is composed of almost one million datapoints, consisting of six circular trajectories, six lemniscate trajectories and random trajectories, generated interpolating randomly generated points in the space. Each of these trajectories is generated with a fixed velocity per trajectory ranging from 1 to $8m/s$. One tenth of the data was held out for testing and parameter tuning.

Real Sensor Noise $\mathbf{v}^{(0)}$	0.01	0.05	0.1
Gal et al. [3]	0.67	0.35	0.03
Gast et al. [6]	0.74	0.37	0.04
Kendall et al. [17]	**0.99**	0.3	-0.11
Ours ($\hat{\mathbf{v}}^{(0)} = 0.01$)	0.95	**0.41**	**0.12**

TABLE S1
LOG-LIKELIHOOD (LL) SCORE FOR INCREASING SENSOR NOISE. HIGHER IS BETTER.

VIII. SENSITIVITY TO SENSOR NOISE ESTIMATES

One of our framework's input consists of the sensor noise variance $\mathbf{v}^{(0)}$, which is propagated through the CNN to recover data uncertainty on output predictions. The value of $\mathbf{v}^{(0)}$ is usually available from the sensor data sheet, or estimated via system identification. In this section, we study the sensitivity of our framework to the precision of the $\mathbf{v}^{(0)}$ estimates. In order to do so, we perform a controlled experiment for the task of steering angle prediction, where each image is corrupted by known Gaussian noise $\mathcal{N}\left(0; \mathbf{v}^{(0)}\right)$. In this experiment, our framework has an estimate of the noise variance $\hat{\mathbf{v}}^{(0)}$, which does not necessarily coincides with the real $\mathbf{v}^{(0)}$. Specifically, we keep $\hat{\mathbf{v}}^{(0)} = 0.01$, while we let $\mathbf{v}^{(0)}$ change. The results of this evaluation are reported in Table S1. Perhaps unsurprisingly, performance peaks when the assumed sensor noise coincides with the real input noise magnitude. However, as the difference between the real and assumed noise variance increases, performance gracefully drops for our approach, indicating the robustness of our method to wrong sensor noise estimates. Interestingly, Table S1 also shows that our approach deals better than the baselines to increasing magnitudes of the noise. This is due to the coupling of data and model uncertainty enforced by our framework.