Fail-Safe Execution of Deep Learning based Systems through Uncertainty Monitoring

Michael Weiss  
*Università della Svizzera italiana*  
Lugano, Switzerland  
michael.weiss@usi.ch  

Paolo Tonella  
*Università della Svizzera italiana*  
Lugano, Switzerland  
paolo.tonella@usi.ch  

Abstract—Modern software systems rely on Deep Neural Networks (DNN) when processing complex, unstructured inputs, such as images, videos, natural language texts or audio signals. Provided the intractably large size of such input spaces, the intrinsic limitations of learning algorithms and the ambiguity about the expected predictions for some of the inputs, not only there is no guarantee that DNN's predictions are always correct, but rather developers must safely assume a low, though not negligible, error probability. A fail-safe Deep Learning based System (DLS) is one equipped to handle DNN faults by means of a supervisor, capable of recognizing predictions that should not be trusted and that should activate a healing procedure bringing the DLS to a safe state.

In this paper, we propose an approach to use DNN uncertainty estimators to implement such supervisor. We first discuss advantages and disadvantages of existing approaches to measure uncertainty for DNNs and propose novel metrics for the empirical assessment of the supervisor that rely on such metrics. We then describe our publicly available tool UNE\textsc{certainty\_wizard}, which allows transparent estimation of uncertainty for regular \textsc{tf.keras} DNNs. Lastly, we discuss a large-scale study conducted on four different subjects to empirically validate the approach, reporting the lessons-learned as guidance for software engineers who intend to monitor uncertainty for fail-safe execution of DLS.

Index Terms—fault tolerance, software reliability, software testing, art neural networks

I. INTRODUCTION

Deep neural networks (DNNs) are a powerful tool to identify patterns in large amounts of data and to make predictions on new, previously unseen data. Thanks to the increased hardware capabilities, DNNs can be run even on small, battery powered hardware and can be trained in performance-optimized GPUs. Correspondingly, the use of DNNs has gained a lot of popularity in the last decade. Moreover, the introduction of high level APIs such as \textsc{tf.keras} (see tensorflow.org) allows even software engineers without previous experience in artificial intelligence to define, train and use custom DNNs. DNNs are now used in many Deep Learning based Systems (DLS), like self driving cars, to interpret observed sensor measurements and control the car’s actuators, in medical systems, to support physicians to make their diagnosis, and in web services, for image processing and analysis.

Relying solely on the predictions made by a deep learning component might be dangerous, as there is always some uncertainty about the correctness of the prediction. In fact, the contract between the overall system and its DNN based components is necessarily a probabilistic one, and while the probability of an error can be low, it is never zero.

The uncertainty intrinsic with DNNs is either caused by entropy in the input or by inadequate training. While the first type of uncertainty is inherent to a problem and cannot be avoided by definition, the latter cannot also be avoided for practical reasons: in most applications, the input space consists of a huge number of input contexts (e.g., the different weather or light conditions in which a car is driven), and it is impossible to collect data which perfectly represents all of them.

Faced with a problem for which a prediction is subject to high uncertainty, a human intelligence may consider to refuse to make a prediction and instead say ‘I do not know’. DNNs on the other hand, will calculate a prediction for any given input, independently of the uncertainty of the prediction. If such predictions are trusted by a DLS, the DLS may fail due to a wrong prediction, as is best illustrated by the following two examples: a self-driving car has recently crashed into an overturned truck in June 2020. A likely explanation for such a crash is that overturned trucks are not sufficiently represented in the cars training data. [1] Second, an online photo storage service classified an image of a black person as a a picture of a gorilla, leading to negative press, which deemed the service as racist. Again, such error is likely caused by insufficient training data of the machine learning component which classified the image. [2] The fact that problems like these happen even in software from leading companies in the machine learning domain shows that preventing such errors is quite challenging.

Even more so, in the second example the solution put in place was a drastic workaround: the label Gorilla was removed from the set of possible predictions for any input.

We propose that DLS include a supervisor, which monitors...
the DNN uncertainty for any given input at runtime, such that
the system can ignore predictions for high-uncertainty inputs
and can run a safe fallback process instead, such as stopping
the self-driving car at the side of the street or, in the second
element, delegating the classification of the image to a human.

The machine learning community has investigated uncertainty-aware types of DNNs, which support the
deployment of such a supervisor. This paper aims at closing
the gap between uncertainty-aware DNNs and the deployment
of an effective supervisor in a DLS. Specifically, it makes the
following contributions:

Metrics Comparison Description and comparison of the
most investigated uncertainty metrics for DNNs, with a
discussion of their advantages and disadvantages.

Uncertainty Wizard Python library which allows zero-
knowledge, transparent implementation of uncertainty-
aware DNNs.

Evaluation Framework We present existing and propose
novel metrics to evaluate DLS which include a supervisor.

Lessons learned We discuss key findings from our empirical
evaluation of various uncertainty metrics applied to four
different case studies.

II. BACKGROUND

In this section, we discuss the different root causes of DNN
faults which can be understood as types of uncertainty and
define the task of DNN fault prediction as a problem of
uncertainty quantification.

A. Sources of Uncertainty

We distinguish between two types of uncertainty; uncer-
tainty caused by a sub-optimal DNN model and uncertainty
cased by randomness in the prediction target. A detailed
discussion of these types is provided by Kendall et al. [3].

Definition 1 (Epistemic Uncertainty): Epistemic uncertainty
is caused by the sub-optimal training or configuration of the
model.

Epistemic uncertainty is sometimes also referred to as model
uncertainty. There are many possible reasons for epistemic
uncertainty, such as insufficient training data, which does
not represent the entire possible input space, sub-optimal
training hyper-parameters and inadequate DNN architecture.
In theory, epistemic uncertainty could be avoided, provided
good enough training data and optimal model configuration.
However, finding such optimal training configurations is im-
possible in most real world applications, as real input spaces,
mostly the scope of this paper, we consider a less general formulation
of uncertainty quantification, which is sufficient for network
supervision [4]:

Definition 3 (Uncertainty Quantification): Uncertainty
Quantification (UQ) is the task of calculating a scalar metric
strictly monotonically increasing in the likelihood (for classi-
fication tasks) or severity (for regression tasks) of a deviation
between the DNN prediction and the ground truth, given a
particular input and the DNN used to do the prediction.

We are thus limiting our interest to the correctness of the
chosen prediction, as opposed to the distribution of all possible
predictions in the output space. The supervisor will reject
inputs for which the uncertainty is above a certain threshold.
Consistently, we define confidence as the opposite of uncertainty, s.t. a confidence metric is supposed to strictly monotonically decrease in the likelihood or severity of a prediction error.

III. Uncertainty-Aware Neural Networks

While regular DNNs, also called Point-Prediction DNNs (PPNN), predict a single scalar value for every output variable, from the perspective of uncertainty quantification it would be preferable if the DNN calculated a distribution over possible output values, which would allow the extraction of the network’s confidence in the prediction. In this section we discuss the various DNN architectures which allow the calculation of such distributions: first, we provide an overview on Bayesian Neural Networks (BNN), which are the standard approach in the machine learning literature about uncertainty quantification. Second, we discuss other approaches, which typically have some theoretical disadvantages when compared to BNNs, but showed good results in practice.

A. Pure Bayesian Neural Networks

BNNs are neural networks where weights are probabilistic, instead of scalar as in PPNN, and are represented as probability density functions. To train a BNN, first, a prior distribution \( p(\theta) \) over weights \( \theta \) has to be defined. Then, given some data \( D \), the posterior distribution \( p(\theta|D) \), i.e., the trained BNN is inferred using Bayes rule:

\[
p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{\int p(D|\theta)p(\theta)d\theta}{p(D)}
\]  

(1)

Since weights are probabilistic, any output of the BNN is also probabilistic and thus allows a statistically well-founded uncertainty quantification. Besides that, these BNNs in their pure form have other advantages over PPNN, among which: they are robust to overfitting and allow to distinguish between epistemic and aleatoric uncertainty.

BNNs in the pure form presented above quickly become intractable due to the large number of integrations required. There has been a long search for mechanisms which make BNN easier to compute, with papers dating back to the early 1990s. To do so, some approaches make additional assumptions, e.g., a normal distribution of the weights, which may cause the BNN to lose some of the advantages listed above. Still, despite many advances in the field, even recent approaches are considered not to scale sufficiently well and are not yet widely used in practice. Other, more scalable approaches have been shown to outperform Bayesian approaches on practical benchmarks.

B. MC-Dropout based Bayesian Neural Networks

In their very influential paper, Gal et al. proposed to approximate BNNs though regular PPNNs. Many PPNNs use dropout layers for regularization. During training, in a dropout layer each neuron activation can be set to 0 with some probability \( p \). This helps to avoid overfitting and can lead to better model performance in general. In their paper, Gal et al. have shown that a PPNN trained with dropout enabled can be interpreted as a BNN, due to the variation induced by the randomized dropout layers. While traditionally the dropout functionality is disabled at prediction time, a dropout-based BNN keeps the random dropping of neuron activations enabled even during predictions and calculates an output distribution by Monte-Carlo sampling of the results in multiple randomized predictions. This approach is thus often referred to as MC-Dropout.

Despite its high popularity, MC-Dropout is not without criticisms: Osband claimed the inability of MC-Dropout to capture every type of uncertainty and several papers have shown that it can be outperformed by other approaches. Also, despite the fact that “the forward passes can be done concurrently, resulting in constant running time”, concurrent processing may not be possible in resource-constrained environments, making MC-Dropout clearly slower than a single prediction of the corresponding PPNN.

C. Deep Ensemble Neural Networks

Lakshminarayanan et al. proposed a fundamentally different approach to quantify uncertainty called Deep Ensembles, or Ensemble for short, i.e., a collection of multiple atomic models for the same problem, differing only in their initialization and trained independently. For every input, a prediction would be made on every atomic model. The predictive distribution could then be inferred from these samples. While deep ensembles are not inherently BNNs, it is possible to interpret them as BNN after applying some minor changes to the parameter regularization. Nonetheless, even in their plain form, they have been shown to outperform MC-Dropout on the task of uncertainty quantification.

Deep Ensembles are, compared to a single PPNN, slow to train if executed sequentially and memory intensive if used concurrently. This may prevent the use of ensembles in some constrained environments. A variety of improvements and modifications have been proposed for Deep Ensembles (Ilg et al. provide a good overview of them).

D. Point Predictor Classifiers

DNNs used for classification typically have a softmax output layer, including one output neuron per class. The output for each class is between 0 and 1, with the sum of all outputs being exactly 1. These outputs are often interpreted as probability distributions over the classes and are used for network supervision by means of the following quantifiers:

Definition 4 (Max-Softmax (SM)): The highest softmax score is used as confidence quantification (also referred to as Vanilla or Softmax Prediction Probability quantification).

Definition 5 (Prediction Confidence Score (PCS)): The difference between the two highest softmax outputs is used as confidence quantification.
Definition 6 (Softmax-Entropy (SME)): The entropy over the outputs of the softmax layer is used as confidence quantification. These quantifiers are often criticized as a poor approach to uncertainty quantification. As opposed to BNNs and BNN approximations, PPNN based quantifiers are not theoretically well founded, and can be shown to severely overestimate a network’s confidence \cite{10}. As a further disadvantage, such a PPNN based approach can not be directly applied to regression problems.

E. Inferring Prediction and Uncertainty from Samples

In MC-Dropout and Deep Ensembles, samples need to be aggregated into a point prediction and uncertainty quantification, and the literature provides a variety of quantifiers able to do so. In this Section, we first discuss the quantifiers applicable to regression problems, and then the ones for classification problems.

a) Regression Problems: In their proposition of MC-Dropout, Gal et al. \cite{10} propose to use the average of the observed samples as prediction, and their predictive variance as uncertainty:

Definition 7 (Predictive Variance): The *predictive variance* is the sample variance over the observed samples plus the inverse model precision. The inverse model precision is constant for a given model. Thus, for the purpose of network supervision, where strictly monotonic transformations of uncertainty quantification scores do not change the supervisor performance, using predictive variance is equivalent to using the sample variance or the sample standard deviation. An alternative approach was proposed by Lakshminarayanan et al. \cite{13}. For their deep ensembles, they propose that the atomic models are adapted s.t. they have a second output variable for every regression output which predicts the variance \cite{17}. The uncertainty of the ensemble can then be quantified by averaging these variances.

b) Classification Quantifiers: The following quantifiers are proposed to derive an overall prediction and uncertainty:

Definition 8 (Mean-Softmax (MS)): The overall prediction is the class with the highest sum of softmax scores over all samples and the corresponding confidence is the average softmax score of this class over all samples.

<table>
<thead>
<tr>
<th>Uncertainty-aware DNN</th>
<th>Classification</th>
<th>Regression</th>
<th>BNN</th>
<th>Custom Layers</th>
<th>Training Effort</th>
<th>Prediction Effort</th>
<th>Main Advantage</th>
<th>Main Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variants of pure BNN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Weight</td>
<td>High or intractable*</td>
<td>High or intractable*</td>
<td>Theoretically well founded</td>
<td>Custom architecture, computationally hard*</td>
</tr>
<tr>
<td>MC Dropout</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Dropout</td>
<td>Low</td>
<td>High</td>
<td>Fastest BNN approximation</td>
<td>Sampling for predictions is costly</td>
</tr>
<tr>
<td>Deep Ensembles</td>
<td>Yes</td>
<td>Yes</td>
<td>No*</td>
<td>No</td>
<td>High</td>
<td>High</td>
<td>Good practical results, no major architecture requirements</td>
<td>Computationally and storage intensive</td>
</tr>
<tr>
<td>PPNN-Softmax based</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Softmax</td>
<td>Minimal</td>
<td>Minimal</td>
<td>Fast and Simple</td>
<td>Misleading and no regression</td>
</tr>
</tbody>
</table>

TABLE I: Overview of popular uncertainty-aware DNN techniques (* = approach dependent)

```python
# Use as a regular tf.keras model
model = model.add(tf.keras.layers.Dense(100))
model.compile( ... )
model.fit( ... )
# Use as standard (non-wizard) model
regular_nn_output = model.predict(x_test)
# Predict as Point-Predictor w/ confidence
pred_pp, pcs = model.predict_quantified(x_test, quantifier='pcs')
# Predict as Bayesian Model w/ uncertainty
pred_b, var_r = model.predict_quantified(x_test, num_samples=100, quantifier='var_ratio')
```

Listing 1: Keras-Syntax Stochastic Model

```python
# Use as a regular tf.keras model
model.add(tf.keras.layers.Dense(100))
model.compile( ... )
model.fit( ... )
# Use as a simple tf.keras model
model = wizard.models.StochasticSequential()
# Use the wizard models as point-predictor
model = wizard.models.StochasticSequential()
# as a bayesian model
model = model.add(tf.keras.layers.Dropout(0.2))
model.compile( ... )
model.fit( ... )
```

Listing 2: Convert Pre-Trained Model

MS has been proposed and is often used with Deep Ensembles. It is thus also called *ensembling* [5]. Three other quantifiers have been proposed to be used with MC-Dropout \cite{10}, \cite{18}, \cite{19}: *Variation Ratio (VR)*, which is defined as the percentage of samples for which the overall chosen class is not the class with the highest softmax output; *Predictive Entropy (PE)*, which measures the average amount of information in the predicted distributions; and, *Mutual Information (MI)* between a prediction and the models posterior (see Gal, 2016 \cite{10}, Section 3.3.1 for a precise description and for examples of these three uncertainty quantifiers).

IV. UNCERTAINTY WIZARD

With 148'742 stars and 82'784 forks on github.com, Tensorflow is presumably the most popular deep learning framework. In its recent versions, a large part of its API, *tf.keras*, is based on the popular Keras API, a simple yet powerful high-level API to develop, train and deploy DNNs. The simplicity of the *tf.keras* API allows researchers and practitioners outside of the machine learning community to get

3Its main alternative, *PyTorch* has 42'621 stars and 11'102 forks.
started with deep learning easily. Unfortunately, such simple API does not expose equally simple methods to quantify uncertainty. Thus, we release **Uncertainty Wizard**, an extension of *tf.keras* which allows developers to easily create Stochastic and Ensemble DNNs and apply all uncertainty quantifiers described in Section III. The core features of *Uncertainty Wizard* are:

**Sequential API** Sequential models are the most straightforward way to use *tf.keras* and are thus very popular. However, the sequential API does not allow dropout at prediction time. Hence, it also does not allow the implementation of MC-Dropout. **Uncertainty Wizard** closes this gap by supporting the creation of stochastic models using the sequential, as well as the functional, API in plain *tf.keras* syntax. An example of this is given in [Listing 1](#).

**Dynamic Randomness** In *tf.keras*, the dropout behavior at prediction time is unchangeable for a given model: Either it is disabled (as required in point predictors) or enabled (as required in stochastic models). Thus, despite relying on the same architecture and weights, a *tf.keras* model cannot be used both as stochastic model and as point predictor. Converting them is nontrivial and includes the creation of a new model, which is memory and performance intensive. **Uncertainty Wizard**’s sequential models dynamically enable and disable stochastic behavior based on whether the passed quantifier expects a point prediction or sampled predictions. This is shown in [Listing 1](#) as well.

**Conversion from Keras** Often, the user of a DNN is not the same person or group that trained the model. To allow users to use such a model for MC-Dropout nonetheless, **Uncertainty Wizard** supports the import of any *tf.keras* model (which has at least one dropout layer) as a stochastic model.

**Parallelized Ensembles** Keras does not natively support parallel training of multiple models. Ensembles would thus have to be trained sequentially. Especially with smaller models, which do not require the full use of the existing hardware to be loaded and executed, this has a large negative impact on training and prediction time. Additionally, sequential processing of kers models can lead to pollution of the global tensorflow runtime due to memory leaks and eager processing. **Uncertainty Wizard** treats ensembles lazily: every atomic model is stored on the file system, and lazily loaded into its own tensorflow runtime during execution. This allows faster, parallelized execution without runtime pollution.

**Dependency-Light** *pip install* **Uncertainty Wizard** is platform independent, importable through *pip install uncertainty-wizard* and has only one dependency: Tensorflow version 2.3.0 or later.

Due to space constraints, the description of **Uncertainty Wizard** at this place is brief. **Uncertainty Wizard** and a comprehensive user guide can be found online: [github.com/testingautomated-usi/uncertainty-wizard](https://github.com/testingautomated-usi/uncertainty-wizard)

### V. Supervised Neural Network Assessment

Network supervision can be viewed as a binary classification task: **malicious samples**, i.e., inputs which lead to a misclassification (for classification problems) or to severe imprecision (in regression problems) are positive samples that have to be rejected. Other samples, also called **benign samples**, are negative samples in the binary classification task. An uncertainty based supervisor accepts an input *i* as a benign sample if its uncertainty *u(i)* is lower than some threshold *t*. The choice of *t* is a crucial setting, as a high *t* will fail to reject many malicious samples (false negatives) and a low *t* will cause too many false alerts (false positives).

Differently from standard binary classification tasks, the choice of *t* for network supervision cannot rely on the optimization of an aggregate metric that accounts for both false positives and false negatives, such as the F1-metric, because negative samples are either completely unknown at training time, or, in case some negative samples are known, they cannot be assumed to be representative of all unknown execution conditions that will give raise to uncertainty at runtime. Hence, the choice of *t* is solely based on the false positives observed in the validation set. In practice, given the uncertainties measured on the validation set, *t* shall ensure that at runtime under similar conditions only an acceptable false positive rate *ε* is expected to occur [21].

Existing metrics allow the individual assessment of the supervisor’s performance separately from the assessment of the model performance [22]. However, such measurements do not take into account the interaction between the two: since the output of the model is not used by the DLS when the supervisor activates the fail safe procedure (*u(i) ≥ t*), it does not make any sense to evaluate the performance of the model in such scenarios. For this reason, we propose a new approach for the joint assessment of model and supervisor, which we call *supervised metrics*. In the next two sections we first summarize the state of the art metrics for the separate, individual assessment of model and supervisor, followed by a description of our proposal of a new joint assessment approach.

#### A. Existing Metrics for the Individual Assessment of Model and Supervisor

There are well established metrics for the individual assessment of performance of a model *m*. These are based on some objective function *obj(I,m)*, such as accuracy (ACC) (for classifiers) or mean-squared error (MSE) (for regression models), computed on a test dataset *I*.

Classically, the supervisor’s performance would be assessed individually using performance metrics designed for binary classifiers. For a given *t*, the available metrics include true

---

4See e.g. tensorflow issues [33030](https://github.com/tensorflow/tensorflow/issues/33030) and [37505](https://github.com/tensorflow/tensorflow/issues/37505)
positive rate (TPR), false positive rate (FPR), true negative rate (TNR) and false negative rate (FNR), $F_1$ score and accuracy (ACC). To use these metrics with regression problems, an acceptable imprecision would have to be defined, allowing to divide inputs into benign (negative cases) and malicious (positive cases). Alternatively, the effect of the predictions on the overall DLS system could be monitored to only treat inputs leading to system failures as malicious ones \cite{21}.

There are also existing, classical metrics to assess a binary classifier independently of the threshold $t$. For instance, the average precision score (AVGPR) computes the average of the precision values obtained when varying $t$, weighted by the recall measured at $t$. Another popular threshold independent metric is the area under the receiver operating characteristic curve (AUROC) \cite{15, 19, 21}. When assessing individually the performance of a supervisor, AVGPR should be preferred over AUROC as, amongst other advantages \cite{23}, it is better suited for the unbalanced datasets \cite{24} typically observed during malicious input detection. Threshold independent analysis of the supervisor in a regression problem is straightforward and can be done using point-biserial correlation between the quantified uncertainty and the observed prediction error, given some objective function (e.g. MSE).

Independent analysis of model’s performance, considered in isolation without supervision, and of the supervisor’s performance, again in isolation, results in measurements that do not capture the overall goal of the interaction between supervisor and model under supervision: ensuring high model performance on the samples considered safe by the supervisor, while keeping the amount of samples considered unsafe as small as possible. To capture such goal, we propose novel metrics for joint model and uncertainty quantification assessment in a supervised DLS.

**B. Supervised Metrics for the Joint Assessment of Model and Supervisor**

When considering model and supervisor jointly, we can still use the objective function used to assess the model in isolation, but we evaluate it in a supervised context: given a test set $I$ and an objective function $\text{obj}(I, m)$ for model $m$ with uncertainty quantifier $u$ and supervisor threshold $t$, the supervised objective function $\overline{\text{obj}}(I, m)$ is defined as:

$$\overline{\text{obj}}(m, I) = \text{obj}\{\{i \mid i \in I \text{ and } u(i) < t\}\}$$

i.e., the objective is applied only to the subset of inputs which is accepted by the supervisor. By decreasing $t$, assuming that the cardinality of the resulting subset of inputs remains big enough to calculate a statistically significant $\overline{\text{obj}}(I)$, we may generally get higher and higher values of the supervised objective function $\overline{\text{obj}}(I)$. However, such high values of $\overline{\text{obj}}(I)$ are likely associated with a high false alarm rate of the supervisor. Thus, any $\overline{\text{obj}}(I)$ should always be regarded in conjunction with the acceptance rate of the supervisor $\Delta_u(I)$:

$$\Delta_u(I) = \frac{|\{i \mid i \in I \text{ and } u(i) < t\}|}{|I|}$$

Similar to the popular $F_1$ score, the following combination of these two metrics allows to capture the effectiveness of the collaboration between supervised model and supervisor:

**Definition 9 (S-Score):** The $S_1$-Score measures the harmonic mean of a supervised objective function, normalized between zero and one, and the supervisors acceptance rate as

$$S_1(m, u, I) = \frac{2}{\overline{\text{obj}}(I, m) - \text{obj}^-} + \Delta_u(I)^{-1}$$

where $\text{obj}^-$ and $\text{obj}^+$ are the lower and upper bounds used for normalization. For classifiers, if accuracy is the objective function, $\text{obj}^- := 0$ and $\text{obj}^+ := 1$. For regression problems, or more generally for unbounded objective functions, $\text{obj}^-$ and $\text{obj}^+$ have to be estimated empirically (e.g., based on the empirical distribution of the objective function values), independently from $m$ and $u$.

The $S_1$ scores weights $\Delta_u(I)$ and $\overline{\text{obj}}(m, I)$ equally. Equivalent to the popular $F_1$ score, other $S_\beta$ scores can be used, where $\beta > 0$ is the weighting parameter \cite{25}.

$$S_\beta(m, u, I) = (1 + \beta^2) \cdot \frac{\overline{\text{obj}}(I, m) - \text{obj}^-}{\overline{\text{obj}}(I, m) - \text{obj}^+} \cdot \Delta_u(I)$$

**VI. CASE STUDIES**

We assess the uncertainty quantification capabilities of point predictors, deep ensemble and MC dropout using different quantifiers. We intentionally focus our study on uncertainty quantifiers which can be applied to traditional and widely used DNN architectures, and we exclude those implementing the pure Bayesian form of uncertainty estimation, since they require the adoption of dedicated architectures where the network weights encode a probability distribution, not just a scalar value. This restriction, in combination with UNCERTAINTY WIZARD, allows developers to measure uncertainty at minimal effort, given a traditional DNN. The goal of our empirical evaluation is to assess the usefulness of the uncertainty quantifiers supported by UNCERTAINTY WIZARD when used as supervisors, as well as to collect lessons learned that practitioners can follow when applying UNCERTAINTY WIZARD to their DNNs.

**A. Research Questions**

We consider the following research questions:

**RQ1 (effectiveness):** How effective are supervisors at increasing the supervised model’s performance?

This is the key research question of our empirical study, since the main hypothesis behind supervisors is that they can prevent usage of a model when its performance is predicted to be low. Hence, we expect an increase of the supervised model’s performance $\overline{\text{obj}}$ as compared to the unsupervised one $\text{obj}$.

**RQ2 (comparison):** Is there a supervisor and quantifier type which yield optimal performance across subjects and across alternative choices of the uncertainty threshold?

We consider three types of uncertainty estimators, Point Predictors, MC-Dropout and Ensemble, and several uncertainty
quantifiers, respectively [SM, PCS, SME], [VR, PE, MI, MS] (see Section III). We want to investigate whether any combination of estimator type and quantifier dominates all the others in terms of $S$-score. To investigate how performance changes with the uncertainty threshold $t$, we consider different acceptable rates $\epsilon$ of false positives on the nominal data and we compute the threshold $t$ that ensures such false positive rates on the validation set, so that we can compare alternative estimators/quantifiers at equal FPR on the validation set of the nominal data.

**RQ3 (sample size): How many samples are required in stochastic and ensemble models to get reliable uncertainty quantification?**

Since the main cost associated with the usage of MC-Dropout and Ensemble is the collection of multiple samples for each individual prediction, we want to understand what is the minimum sample size that ensures good performance of each different supervisor. In particular, we study the convergence of supervised accuracy to its asymptotic value as the sample size is increased.

**RQ4 (sensitivity): How sensitive are supervisors to changes in their main hyperparameters?**

With this research question we want to understand whether the choice of hyperparameters is critical to get optimal performance, or on the contrary if they can be chosen in the neighbourhood of the optimal choice with minor impact on the resulting performance of the supervisor. For Point Predictors, we consider the number of training epochs as the main hyperparameter; for MC-Dropout, the number of training epochs and the number of samples; for Ensemble, the number of training epochs and the number of atomic models. We measure the standard deviation of the supervised objective function (e.g., supervised accuracy) in the neighbourhood of the optimal choice with minor impact on the resulting performance of the supervisor.

### Table II: Overview of supervision capabilities of different techniques, i.e., model types and quantifier combinations for different thresholds.

<table>
<thead>
<tr>
<th>Technique</th>
<th>$\epsilon = 0.1$</th>
<th>$\epsilon = 0.01$</th>
<th>$\epsilon = 0.1$</th>
<th>$\epsilon = 0.01$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SM</td>
<td>ACC 0.82</td>
<td>S-C 0.67</td>
<td>$\Delta_u 0.90$</td>
<td>$S_1 0.80$</td>
</tr>
<tr>
<td>PCS</td>
<td>ACC 0.82</td>
<td>S-C 0.71</td>
<td>$\Delta_u 0.90$</td>
<td>$S_1 0.80$</td>
</tr>
<tr>
<td>SME</td>
<td>ACC 0.82</td>
<td>S-C 0.84</td>
<td>$\Delta_u 0.88$</td>
<td>$S_1 0.80$</td>
</tr>
<tr>
<td>VR</td>
<td>ACC 0.82</td>
<td>S-C 0.84</td>
<td>$\Delta_u 0.88$</td>
<td>$S_1 0.80$</td>
</tr>
<tr>
<td>PE</td>
<td>ACC 0.82</td>
<td>S-C 0.84</td>
<td>$\Delta_u 0.88$</td>
<td>$S_1 0.80$</td>
</tr>
<tr>
<td>MI</td>
<td>ACC 0.86</td>
<td>S-C 0.87</td>
<td>$\Delta_u 0.92$</td>
<td>$S_1 0.80$</td>
</tr>
<tr>
<td>MS</td>
<td>ACC 0.86</td>
<td>S-C 0.87</td>
<td>$\Delta_u 0.92$</td>
<td>$S_1 0.80$</td>
</tr>
<tr>
<td>SME</td>
<td>ACC 0.82</td>
<td>S-C 0.84</td>
<td>$\Delta_u 0.88$</td>
<td>$S_1 0.80$</td>
</tr>
<tr>
<td>PCS</td>
<td>ACC 0.82</td>
<td>S-C 0.84</td>
<td>$\Delta_u 0.88$</td>
<td>$S_1 0.80$</td>
</tr>
<tr>
<td>SM</td>
<td>ACC 0.86</td>
<td>S-C 0.87</td>
<td>$\Delta_u 0.92$</td>
<td>$S_1 0.80$</td>
</tr>
<tr>
<td>SME</td>
<td>ACC 0.82</td>
<td>S-C 0.84</td>
<td>$\Delta_u 0.88$</td>
<td>$S_1 0.80$</td>
</tr>
<tr>
<td>VR</td>
<td>ACC 0.82</td>
<td>S-C 0.84</td>
<td>$\Delta_u 0.88$</td>
<td>$S_1 0.80$</td>
</tr>
<tr>
<td>PE</td>
<td>ACC 0.82</td>
<td>S-C 0.84</td>
<td>$\Delta_u 0.88$</td>
<td>$S_1 0.80$</td>
</tr>
<tr>
<td>MI</td>
<td>ACC 0.86</td>
<td>S-C 0.87</td>
<td>$\Delta_u 0.92$</td>
<td>$S_1 0.80$</td>
</tr>
<tr>
<td>MS</td>
<td>ACC 0.86</td>
<td>S-C 0.87</td>
<td>$\Delta_u 0.92$</td>
<td>$S_1 0.80$</td>
</tr>
</tbody>
</table>

**B. Subjects**

We use the following classification problems as case study subjects, aiming to increase diversity and practical relevance.

**Mnist [26]** Classification of handwritten digits, formatted as small grayscale images. This is the most popular dataset in machine learning testing [4], and a relatively easy problem, where even simple models achieve high accuracy. We took the DNN architecture from a Keras tutorial [27].

---

7
Cifar10 [28] Classification of colored images into ten different classes. It is also very popular in DLS testing [4] and it represents a more challenging task than Mnist. We use the model architecture proposed in the Brownlees Cifar10 tutorial [29].

Traffic [30] Classification of images of European traffic signs [31]–[38]. The different sources the data was collected from, combined with the fact that the dataset is unbalanced and many images are of bad quality, reflect a quite realistic, high-uncertainty setup. Since traffic sign recognition is a core component of self-driving cars, this is also a very interesting case study from the software and system engineering point of view. The model architecture we used was proposed alongside the release of this dataset [30].

Imagenet [39] (Pretrained) Image classification problem with as many as 1,000 classes. We use eight pre-trained Efficientnet models [40]. As for this subject we rely on pre-trained models (which include dropout layers), we can test them only as MC-Dropout and Point Predictor models, but not as Ensembles.

C. Experimental Setup

Except for the pre-trained ones, models were trained for 200 epochs. After every epoch, we assessed the models’ performance on both a nominal and an out-of-distribution (OOD) dataset, for every quantifier. To do so, we used Mnist-c [41] as OOD test set for Mnist and the color-image transformations proposed by Hendrycks [42] to generate OOD samples for the other subjects. We used three different thresholds, calculated on the nominal validation set to ensure the lowest possible FPR above $\epsilon$, with $\epsilon \in \{0.1, 0.3, 0.5\}$ respectively. To measure the sensitivity to the number of samples, quantifiers of deep ensembles were assessed with every number of atomic models between 2 and 50, Similarly, MC-Dropout was assessed on every number of samples between 2 and 100.

Counting atomic models individually, this procedure required the training of 153 DNNs, and the calculation of $2'121'600$ DNN predictions. Due to the high workload, the training and prediction processes were distributed on three different workstations using Windows or Ubuntu and four different GPU types (one workstation had two GPUs). Uncertainty Wizard was used for training, prediction and uncertainty quantification.

D. Results

We organize the analysis of the results obtained in our experiments by research question.

1) RQ1 (Effectiveness): An overview of our results is provided in Table II. Due to space constraints, the results for $\epsilon = 0.05$ are omitted in the table and the values for the 8 different ImageNet models are averaged. The full set of results can be found in the online replication package.

Our results suggest that all supervisors lead to supervised accuracies $\textit{ACC}$ which are at least as high, but typically much higher than the accuracy $\textit{ACC}$ of the unsupervised model. Thus, supervisors are effective. The effectiveness is particularly strong on the OOD datasets: For example, on Mnist, where the unsupervised point predictor has an accuracy of 74%, the supervised accuracy at $\epsilon = 0.1$, is above 95% with most supervisors. In other words, a DLS using an unsupervised model will experience six times more faulty predictions than the unsupervised one. Also notable are the results on the nominal Mnist dataset, where even simple supervisors based on point predictors turn an unsupervised accuracy of 96% (at $\epsilon = 0.1$) into that of a nearly perfect predictor while still accepting around 88% of the inputs.

Summary (RQ1): All tested supervisors are, in general, effective at increasing the supervised accuracy compared to the unsupervised accuracy.

2) RQ2 (Comparison): If we look at the $S_1$-Score in Table II, it is apparent that there is no uncertainty quantifier which outperforms all the other ones on every subject/dataset and for every threshold ($\epsilon$). To allow for an overall comparison of the quantifiers, we computed the average ranks of the quantifiers when ordered by $S_1$-Score. Results are shown in Table III where $N$ indicates the number of data points on which average ranks have been computed. While there is no absolute dominant supervisor, i.e., the ideal choice of supervisor remains problem dependent, Ensembles can be considered the overall best performing supervisors (in line with existing literature [6]), because when they do not have the lowest rank, they still have quite low rank values. Actually, even without supervision Ensembles often achieve higher accuracy than Point Predictors and MC-Dropout based models. Interestingly, in most cases the sophisticated quantifiers PE and MI, grounded on information theory, do not perform better and often perform worse than the simpler quantifiers VR and AS. Despite the theoretical disadvantages of Point Predictors, in our experiments this simple approach outperformed the

<table>
<thead>
<tr>
<th>Technique</th>
<th>Per Subject</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mnist</td>
<td>traffic</td>
</tr>
<tr>
<td></td>
<td>N=6</td>
<td>N=6</td>
</tr>
<tr>
<td>Point Pred</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SM</td>
<td>3.33</td>
<td>9.67</td>
</tr>
<tr>
<td>PCS</td>
<td>2.67</td>
<td>9.83</td>
</tr>
<tr>
<td>SME</td>
<td>5.67</td>
<td>10.0</td>
</tr>
<tr>
<td>VR</td>
<td>5.83</td>
<td>5.83</td>
</tr>
<tr>
<td>PE</td>
<td>7.00</td>
<td>6.67</td>
</tr>
<tr>
<td>MI</td>
<td>6.67</td>
<td>6.83</td>
</tr>
<tr>
<td>MS</td>
<td>5.33</td>
<td>7.17</td>
</tr>
<tr>
<td>VR</td>
<td>4.17</td>
<td>3.17</td>
</tr>
<tr>
<td>PE</td>
<td>8.17</td>
<td>1.33</td>
</tr>
<tr>
<td>MI</td>
<td>10.17</td>
<td>3.0</td>
</tr>
<tr>
<td>MS</td>
<td>7.0</td>
<td>2.5</td>
</tr>
</tbody>
</table>

TABLE III: Rank-Order Analysis: $S_1$-Score ranks, averaged over $\epsilon \in \{0.01, 0.05, 0.1\}$, nominal and OOD datasets.
Fig. 2: Influence of sample size in MC-Dropout (a) and number of atomic models in Ensemble (b) on supervised accuracy $\text{acc}$; values taken with $\epsilon = 0.1$ and MS quantifier on nominal dataset.

3) RQ3 (Sample size): We find that for both MC-Dropout and Ensembles the relatively low number of 20 samples is already sufficient to get a similar supervised accuracy as with a much higher number of samples. This is shown in Figure 2 for the MS quantifier and $\epsilon = 0.1$. 20 samples, while still higher than what’s recommended in related literature [6], is probably small enough to be used in practice in many applications. The other quantifiers behave similarly, with one notable exception, which is the VR quantifier: VR can only take a finite set of discrete values, which can be easily shown to be equal to the number of samples. Hence, a low sample size makes it impossible to set thresholds well fit to the target $\epsilon$, because thresholds are correspondingly also discrete and limited to the number of samples. So, to achieve the target FPR $\epsilon$ precisely, we might need substantially more than 20 samples.

Summary (RQ3): A few (~20) samples are enough to get good supervision results with most quantifiers (VR represents an exception, due to the discretization of the values it can take).

4) RQ4 (Sensitivity): The number of training epochs and the number of samples/atomic models can be visualized as a 200 (number of epochs) by 100 (number of samples) or 50 (number of atomic models) grid. To assess the sensitivity of supervisors, we calculate the standard deviation ($\text{std}$) of $\text{ACC}$ within a 5x5 filter applied to this grid. In this way, we account for the variability of the supervised accuracy in a 5x5 neighbourhood of each hyperparameter configuration. The higher the std, the higher the sensitivity to small hyperparameter changes in the neighbourhood. We also calculate the average $\text{ACC}$ in such 5x5 neighbourhood. The result is a pair of heatmaps, an example of which is shown in Figure 3. This figure suggests that hyperparameter sensitivity negatively correlates with $\text{ACC}$: Hyperparameters leading to low accuracy (bright colors in Figure 3a) typically show high sensitivity to hyperparameter changes (dark colors in Figure 3b). We do indeed observe this negative correlation for all case studies. The values of point-biserial correlation, shown in the $S-C$ (Sensitivity-Correlation) columns of Table II, are strongly negative in most cases, with $p$-value < 0.05 in 358 out of 390 cases. In low accuracy cases, small changes to number of samples and number of epochs have a high effect on accuracy.

Summary (RQ4): For a given supervisor and model, for what concerns the number of training epochs and samples used for quantification, the higher the supervised accuracy, the lower the model’s sensitivity to small hyperparameter changes.

E. Lessons Learned

Based on our answers to the RQs, we distilled the following primary lessons learned, possibly useful for a practical usage of uncertainty-quantifiers based uncertainty monitoring for DLS robustness:

Anything is better than nothing: While the selection of the ideal supervisor is problem dependent, all the supervisors we tested showed some capability to increase the accuracy of the DNN when supervised. Thus including any uncertainty monitoring supervisor in a DLS increases its fail-safety.
Ensembles are powerful: Not only did Ensembles show the best average rank on the $S_1$ score (ignoring the pre-trained Imagenet studies), in many cases even the unsupervised accuracy increased. Furthermore, the relatively low number of 20 atomic models was sufficient to achieve good results in our study. Thus, provided sufficient system resources, we suggest software architects to use Ensembles instead of Point Predictors. On the other hand, the latter may represent a good compromise solution when computational resources are severely constrained.

Number of samples affects choice of quantifier: For Ensembles and MC-Dropout, VR was the best quantifier on average, but it requires a large number of samples to allow for precise threshold selection. Thus, if computational resources allow the calculation of a high number of samples, our experiments suggest to use VR as quantifier. Otherwise, MS showed good performance, despite its simplicity.

In-production supervisor assessment is needed: Since there is no uncertainty quantifier which performs best in all cases, we want to emphasize the importance of in-production assessment of the supervisors performance on the actual system to be supervised, by comparing different supervisors for optimal selection. The metrics that we propose in Section [V] are specifically designed for such assessment.

**F. Threats to Validity**

External Validity: While we considered only four subjects, we diversified them as much as possible. In particular, besides the benchmark subjects often used in DNN testing (Mnist, Cifar10 and Imagenet), we included an additional subject, Traffic, which consists of unbalanced data, partially of low quality. It implements a functionality (traffic sign recognition) commonly integrated in autonomous vehicles.

Internal Validity: The selection of hyperparameters for DNN training might be critical and the selected values may not be representative of other contexts. To address this threat, we refrained from selecting any hyperparameter for the case study models ourselves, wherever possible, and instead relied on architectures available from the literature. For what concerns the internal hyperparameters of the supervisors, we evaluated the sensitivity of the results to their choice in a dedicated research question (RQ4).

Another threat to the internal validity of our study is that the OOD unexpected inputs used in our experiments might not be representative of the uncertainties that may be observed in practice. Indeed, this is unavoidable and intrinsic to the problem of DNN supervision, as unexpected conditions occurring in practice cannot be by definition simulated ahead of time.

**VII. RELATED WORK**

Empirical Studies of Uncertainty-Aware Deep Neural Networks: Oliveira et al. [43] compared, amongst others, MC-Dropout based uncertainty and a variational approximation of BNN. In their experiments, the performance of the two were comparable, but MC-Dropout was much faster. They did not consider Ensemble models. Similarly, but more extensively, Ovadia et al. [6] compared various uncertainty aware DNNs against each other, including MC-Dropout, Deep Ensembles and a variational approximation of BNN. Consistently with our results, Ensembles performed the best in their experiments. As opposed to our work, both of these studies consider fewer subjects, do not investigate the impact of different quantifiers, and do not have the constraint of transparently introducing uncertainty estimators into DNNs without altering their inner architecture, as possible instead with BNN approximations. Zhang et al. [16] compare MC-Dropout against PCS to detect misclassifications caused by adversarial examples, i.e., examples deliberately modified to trick the DNN into making prediction errors. Their results show, similarly to ours, that there is no strict dominance between these two approaches of network supervision. Ours is the first large scale study where uncertainty estimators are injected transparently into existing DNNs (thanks to UNCERTAINTY WIZARD). We are also the first to introduce practical metrics for in-production assessment of supervisors and to distill a list of lessons learned that can be used as guidelines for practical usage of supervisors.

Other types of DNN supervisors: While our focus is on uncertainty measures based on the variability of the output for a given input, there have been various proposals of supervisors that are not directly based on the output distribution of the supervised network. Berend et al. [15] compared various such techniques, typically based on neuron activations, which recognize activation patterns that were not sufficiently represented in the training data. While such an approach may be powerful against epistemic uncertainty, it cannot help against aleatoric uncertainty. Stocco et al. [21] and Henriksson et al. [22] proposed the use of autoencoders, i.e., anomaly detectors trained on the DLS training set as DNN supervisors. This approach does not consider the inner state of the DNN or its predictions (i.e., it is black-box). On the contrary, the supervisors supported by UNCERTAINTY WIZARD take advantage of the inner state of the DNN being supervised.

**VIII. CONCLUSION**

Despite their fundamental role to support fail-safe execution, uncertainty estimators are rarely used by developers when integrating DNNs into complex DLS. This might be due to the high complexity of some of the approaches and the lack of a tool to facilitate the use of such techniques. This paper closes such gap through the following contributions: (1) We compared the most widely used uncertainty-aware DNN types, providing an easy start into the relevant literature; (2) We released our tool UNCERTAINTY WIZARD, which allows to build and evaluate uncertainty-aware DNNs obtained transparently from unchanged, traditional DNNs; (3) We reported our empirical evaluation, summarized into practical guidelines on how to set up an uncertainty monitoring DNN supervisor for a production system.
The non-optimality of any approach under all conditions may allow a complementary use of multiple supervisors. To this extent, we plan to investigate the combination of different supervisors as future work, hopefully leading to an overall more stable and universally applicable supervision.

REFERENCES


