European Research Council
Established by the European Commission

Self-assessment Oracles for Anticipatory Testing

TECHNICAL REPORT: TR-Precrime-2020-05

Michael Weiss, Paolo Tonella

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

Project no.: 787703
Funding scheme: ERC-2017-ADG
Start date of the project: January 1, 2019
Duration: 60 months

Technical report num.: TR-Precrime-2020-05
Date: September, 2020
Organization: Università della Svizzera italiana
Authors: Michael Weiss, Paolo Tonella
Dissemination level: Public
Revision: 1.0

Università della Svizzera Italiana (USI)

Principal investigator: Prof. Paolo Tonella
E-mail: paolo.tonella@usi.ch
Address: Via Buffi, 13 – 6900 Lugano – Switzerland
Tel: +41 58 666 4848
Project website: https://www.pre-crime.eu/

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 approaches. We then describe our publicly available tool UNCERTAINTY-WIZARD, which allows transparent estimation of uncertainty for regular *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.

Contents

1 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 *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. A likely explanation for such a crash is that overturned trucks are not sufficiently represented in the cars training data. [39] 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. [42] 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 example, delegating the classification of the image to a human.

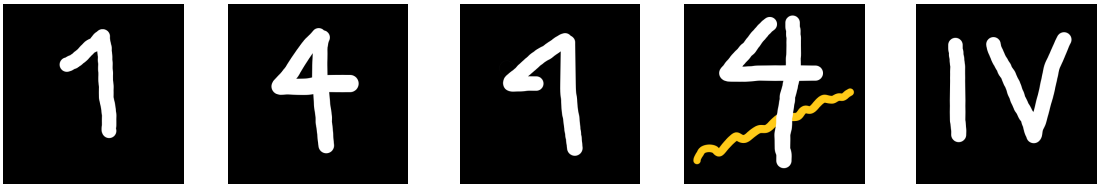
The machine learning community has investigated *uncertainty-aware* types of DNNs, which support the deployment of such a supervisor. This paper aims at closing the gap between uncertainty-aware DNNs and the deployment of an effective supervisor in a DLS. Specifically, it makes the following contributions:

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.



(a) (b) (c) (d) (e)

Figure 1: Examples of uncertainties in digits classification: (a) and (b) cause no uncertainty, (c) causes aleatoric uncertainty, (d) and (e) cause epistemic uncertainty.

2 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.

2.1 Sources of Uncertainty

We distinguish between two types of uncertainty; uncertainty caused by a sub-optimal DNN model and uncertainty caused by randomness in the prediction target. A detailed discussion of these types is provided by Kendall *et al.* [15].

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 and data is impossible in most real world applications, as real input spaces, as well as the space of the possible hyper-parameters and architectural choices, are typically too large.

The second type of uncertainty, which not even an optimal training set and model configuration can avoid, is called *aleatoric uncertainty*:

Definition 2 (Aleatoric Uncertainty) *Aleatoric uncertainty is the uncertainty present in the true (unknown) distribution we are making predictions about.*

Thus, aleatoric uncertainty can be seen as randomness, ambiguity or entropy in the prediction target. When predicting a random event, even an optimal model will make wrong predictions. As aleatoric uncertainty is independent of the model, but instead depends on the predicted data, it is also referred to as *data uncertainty* or *irreducible uncertainty*. Aleatoric Uncertainty can be further distinguished between *homoscedastic uncertainty*, where the uncertainty applies to all data, and *heteorscedastic uncertainty*, where the uncertainty is more prevalent amongst some subsets of the data.

Figure 1 provides a visual example of the difference between aleatoric and epistemic uncertainty. The input to a classifier DNN is the image of a handwritten digit to be recognized. Figures 1a and 1b illustrate regular inputs with low uncertainty. Figure 1c is an example of a figure with high heteroscedastic aleatoric uncertainty: Clearly, the image shows either a 1 or a 4, but it is impossible to say with certainty which one the writer intended. Figure 1d illustrates a common reason for epistemic uncertainty: A small perturbation of the image background, if not present in the training

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

Table 1: Overview of popular uncertainty-aware DNN techniques (* = approach dependent)

data, may lead the model to be incapable of predicting the correct label. Similarly, 1e shows an unexpected input leading to epistemic uncertainty. While the true label is unambiguously 4, a model which was not trained on roman number representations will not be capable to make a correct prediction.

2.2 Uncertainty Quantification

Ideally, instead of predicting a single value as output, a DNN should calculate a probability density function for regression problems or a likelihood for every outcome in a classification problem. As such, every outcome would have its uncertainty quantified (e.g., by the variance of the output probability distribution). We will discuss models capable of calculating such outputs in Section 3. However, for the scope of this paper, we consider a less general formulation of uncertainty quantification, which is sufficient for network supervision [31]:

Definition 3 (Uncertainty Quantification) *Uncertainty Quantification (UQ) is the task of calculating a scalar metric strictly monotonically increasing in the likelihood (for classification 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.

3 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.[1] Second, we discuss other approaches, which typically have some theoretical disadvantages when compared to BNNs, but showed good results in practice. [29]

[1] For a more precise discussion of BNN, we recommend the comprehensive and usage-oriented article by Jospin *et al.* [14].

3.1 Pure Bayesian Neural Networks

BNNs are neural networks in which 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 θ 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:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\int p(D|\theta)p(\theta)d\theta} \tag{1}$$

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 [25] and allow to distinguish between epistemic and aleatoric uncertainty [14].

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 [21, 25]. 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 [13] and are not yet widely used in practice [14]. Other, more scalable approaches have been shown to outperform Bayesian approaches [29] on practical benchmarks.

3.2 MC-Dropout based Bayesian Neural Networks

In their very influential paper[2], Gal *et al.* [8] 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 [35]. 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 [28] 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 [18, 29]. Also, despite the fact that *"the forward passes can be done concurrently, resulting in constant running time"* [8], concurrent processing may not be possible in resource-constrained environments, making MC-Dropout clearly slower than a single prediction of the corresponding PPNN.

3.3 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 [30]. Nonetheless, even in their plain form, they have been shown to outperform MC-Dropout on the task of uncertainty quantification [18, 29].

[2]More than 2300 citations in the four years since its publication, according to Google Scholar.

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.* [13] provide a good overview of them).

3.4 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 *[29] or* Softmax Prediction Probability *[2] quantification).*

Definition 5 (Prediction Confidence Score (PCS) [44]) *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 [8]. As a further disadvantage, such a PPNN based approach can not be directly applied to regression problems.

3.5 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.

Regression Problems In their proposition of MC-Dropout, Gal *et al.* [8] 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.* [18]. 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 [26]. The uncertainty of the ensemble can then be quantified by averaging these variances.

Classification Problems The following quantifiers are proposed to derive an overall prediction and uncertainty:

```
model = wizard.models.StochasticSequential()
# Use as a regular tf.keras model
model.add(tf.keras.layers.Dense(100)
model.add(tf.keras.layers.Dropout(0.2)
model.add(tf.keras.layers.Softmax(10)
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

```
model = ... # load or create plain keras model
model = wizard.models.stochastic_from_keras(model)
# model can now be used as point-predictor
# and as bayesian model, i.e.,
model.predict(...)
model.predict_quantified(...)
```

Listing 2: Convert Pre-Trained Model

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.*

MS has been proposed and is often used with Deep Ensembles. It is thus also called *ensembling* [14]. Three other quantifiers have been proposed to be used with MC-Dropout [7,8,23]: *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 the *Mutual Information (MI)* between a prediction and the models posterior (see Gal, 2016 [8], Section 3.3.1 for a precise description and for examples of these three uncertainty quantifiers).

4 UNCERTAINTY-WIZARD

With 148'742 stars and 82'784 forks on github.com, Tensorflow is presumably the most popular deep learning framework.[3] [9] 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 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 3. 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.

[3] Its main alternative, *PyTorch* has 42'621 stars and 11'102 forks.

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 *tf.keras* API does not expose simple functionality for parallel training of multiple models. Especially with smaller models, which do not require the full use of the existing hardware to be loaded and executed, sequential training of an ensembles atomic models has a large negative impact on training and prediction time. Additionally, it can also lead to pollution of the global tensorflow runtime due to memory leaks and eager processing.[4] 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. We provide a more extensive discussion in a technical tool paper [43]. UNCERTAINTY-WIZARD and a comprehensive user guide can be found online: **github.com/testingautomated-usi/uncertainty-wizard**

5 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 [37].

Existing metrics allow the individual assessment of the supervisor's performance separately from the assessment of the model performance [12]. However, such measurements do not take into

[4] See e.g. tensorflow issues 33030 and 37505.

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 ($u(i) \geq t$), 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.

5.1 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.[5]

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 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 [37].

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)* [2, 23, 37]. When individually assessing the performance of a supervisor, AVGPR should be preferred over AUROC as, amongst other advantages [5], it is better suited for the unbalanced datasets [32] 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.

5.2 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 $obj(I, m)$ for model m with uncertainty quantifier u and supervisor threshold t, the *supervised objective function* $\overline{obj}(I, m)$ is defined as:

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

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 $\overline{obj}(I)$, we may generally get higher values of the supervised objective function $\overline{obj}(I)$. However, such high values of $\overline{obj}(I)$ are likely associated with a high

[5] We assume that obj has to be maximized. The extension to an obj that should be minimized is straightforward. Also note that obj does not have to be the same function used to optimize the DNN during training.

false alarm rate of the supervisor. Thus, any $\overline{obj}(I)$ should always be regarded in conjunction with the acceptance rate $\Delta_u(I)$ of the supervisor:

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

Similar to the popular $F1$ 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*

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

where obj^- and obj^+ are the lower and upper bounds used for normalization of the objective function. For classifiers, if accuracy is the objective function, $obj^- := 0$ and $obj^+ := 1$. For regression problems, or more generally for unbounded objective functions, obj^- and 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 $\overline{obj}(m, I)$ equally. Equivalent to the popular F_1 score, other S_β scores can be used, where $\beta > 0$ is the weighting parameter [41]:

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

6 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.

6.1 Research Questions

We consider the following research questions:

RQ$_1$ (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 $\overline{obj}$ as compared to the unsupervised one obj.

RQ$_2$ (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], [VR, PE, MI,

			Nominal (regular test data)									Out of Distribution (corrupted test data)								
				ϵ = 0.01				ϵ = 0.1					ϵ = 0.01					ϵ = 0.1			
		Technique	ACC	S-C	$\overline{ACC}$	Δ_u	S_1	S-C	$\overline{ACC}$	Δ_u	S_1	ACC	S-C	$\overline{ACC}$	Δ_u	S_1	S-C	$\overline{ACC}$	Δ_u	S_1
Cifar10	Point	SM	0.82	-0.67	0.83	0.97	0.90	-0.67	0.90	0.80	0.85	0.82	-0.47	0.83	0.98	0.90	-0.52	0.89	0.81	0.85
		PCS	0.82	-0.71	0.83	0.97	0.90	-0.70	0.90	0.80	0.85	0.82	-0.49	0.83	0.97	0.90	-0.54	0.89	0.81	0.85
		SME	0.82	-0.64	0.84	0.97	0.90	-0.61	0.91	0.80	0.85	0.82	-0.50	0.83	0.97	0.90	-0.54	0.89	0.80	0.84
	MC-Dropout	VR	0.82	-0.68	0.84	0.98	0.90	-0.68	0.90	0.82	0.86	0.82	-0.63	0.83	0.98	0.90	-0.62	0.89	0.82	0.85
		PE	0.82	-0.70	0.84	0.97	0.90	-0.63	0.90	0.82	0.86	0.82	-0.59	0.83	0.97	0.90	-0.43	0.88	0.83	0.85
		MI	0.82	-0.76	0.84	0.97	0.90	-0.64	0.89	0.82	0.86	0.82	-0.57	0.83	0.98	0.90	-0.54	0.88	0.83	0.86
		MS	0.82	-0.66	0.84	0.97	0.90	-0.67	0.91	0.81	0.86	0.82	-0.56	0.83	0.98	0.90	-0.52	0.89	0.81	0.85
	Ensemble	VR	0.86	-0.66	0.87	0.97	0.92	-0.78	0.93	0.83	0.88	0.87	-0.67	0.87	0.98	0.92	-0.74	0.93	0.83	0.87
		PE	0.86	-0.78	0.87	0.98	0.92	-0.81	0.92	0.84	0.88	0.87	-0.69	0.87	0.98	0.92	-0.63	0.91	0.85	0.88
		MI	0.86	-0.78	0.87	0.98	0.92	-0.83	0.92	0.84	0.88	0.87	-0.68	0.87	0.98	0.92	-0.72	0.91	0.84	0.87
		MS	0.86	-0.73	0.87	0.97	0.92	-0.80	0.94	0.83	0.88	0.86	-0.67	0.87	0.98	0.92	-0.64	0.93	0.83	0.88
Mnist	Point	SM	0.96	-0.80	0.97	0.99	0.98	-0.89	0.99	0.88	0.93	0.74	-0.13	0.84	0.82	0.83	-0.88	0.96	0.48	0.64
		PCS	0.96	-0.79	0.97	0.99	0.98	-0.86	0.99	0.88	0.93	0.74	-0.32	0.80	0.88	0.84	-0.87	0.96	0.49	0.65
		SME	0.96	-0.90	0.97	0.99	0.98	-0.92	1.00	0.88	0.93	0.74	-0.20	0.85	0.79	0.82	-0.90	0.96	0.47	0.63
	MC-Dropout	VR	0.97	-0.63	0.97	0.98	0.98	-0.77	0.99	0.87	0.93	0.74	-0.15	0.82	0.85	0.84	-0.64	0.95	0.51	0.67
		PE	0.97	-0.73	0.97	0.99	0.98	-0.81	1.00	0.87	0.93	0.74	-0.10	0.85	0.79	0.82	-0.81	0.97	0.45	0.62
		MI	0.97	-0.72	0.97	0.98	0.98	-0.87	0.99	0.87	0.93	0.74	-0.15	0.79	0.88	0.83	-0.61	0.91	0.55	0.69
		MS	0.96	-0.71	0.97	0.99	0.98	-0.84	1.00	0.88	0.93	0.74	-0.03	0.84	0.82	0.83	-0.76	0.96	0.47	0.64
	Ensemble	VR	0.97	-0.74	0.98	0.98	0.98	-0.81	0.98	0.95	0.97	0.74	-0.30	0.86	0.78	0.82	-0.75	0.91	0.65	0.76
		PE	0.97	-0.89	0.97	0.99	0.98	-0.93	0.99	0.88	0.93	0.74	-0.34	0.86	0.77	0.81	-0.89	0.97	0.45	0.61
		MI	0.97	-0.82	0.97	0.98	0.98	-0.87	1.00	0.87	0.93	0.74	-0.45	0.87	0.71	0.78	-0.86	0.98	0.42	0.59
		MS	0.97	-0.87	0.97	0.98	0.98	-0.89	1.00	0.88	0.93	0.75	-0.48	0.86	0.79	0.82	-0.91	0.96	0.46	0.63
Traffic	Point	SM	0.81	-0.02	0.90	0.89	0.89	-0.27	0.97	0.75	0.85	0.79	0.02	0.90	0.85	0.88	-0.16	0.97	0.70	0.81
		PCS	0.81	0.04	0.90	0.89	0.89	-0.21	0.97	0.75	0.84	0.79	0.03	0.90	0.85	0.88	-0.20	0.97	0.70	0.81
		SME	0.81	-0.09	0.90	0.89	0.89	-0.38	0.98	0.72	0.83	0.79	0.00	0.90	0.86	0.88	-0.36	0.98	0.66	0.79
	MC-Dropout	VR	0.81	-0.24	0.91	0.87	0.89	-0.42	0.98	0.70	0.82	0.79	-0.40	0.91	0.84	0.88	-0.42	0.98	0.66	0.79
		PE	0.81	-0.18	0.90	0.89	0.89	-0.40	0.98	0.70	0.81	0.79	-0.14	0.89	0.86	0.88	-0.42	0.98	0.65	0.78
		MI	0.81	-0.23	0.90	0.88	0.89	-0.44	0.98	0.70	0.81	0.79	-0.37	0.89	0.87	0.88	-0.44	0.98	0.65	0.78
		MS	0.81	-0.09	0.91	0.87	0.89	-0.38	0.98	0.70	0.81	0.79	-0.19	0.91	0.84	0.88	-0.37	0.98	0.65	0.78
	Ensemble	VR	0.81	-0.56	0.94	0.84	0.89	-0.57	0.97	0.76	0.86	0.80	-0.47	0.93	0.82	0.87	-0.64	0.97	0.73	0.83
		PE	0.81	-0.46	0.92	0.85	0.89	-0.47	0.98	0.69	0.81	0.80	-0.55	0.93	0.82	0.87	-0.51	0.99	0.63	0.77
		MI	0.81	-0.59	0.94	0.84	0.89	-0.46	0.98	0.69	0.81	0.80	-0.64	0.94	0.82	0.87	-0.56	0.99	0.63	0.77
		MS	0.81	-0.47	0.93	0.85	0.89	-0.48	0.98	0.69	0.81	0.80	-0.45	0.93	0.82	0.87	-0.52	0.99	0.63	0.77
Imagent	Point	SM	0.74	n.a.	0.77	0.95	0.85	n.a.	0.84	0.79	0.81	0.50	n.a.	0.61	0.79	0.69	n.a.	0.75	0.54	0.63
		PCS	0.74	n.a.	0.76	0.97	0.85	n.a.	0.84	0.79	0.81	0.50	n.a.	0.55	0.89	0.68	n.a.	0.72	0.57	0.64
		SME	0.74	n.a.	0.76	0.96	0.85	n.a.	0.83	0.80	0.81	0.50	n.a.	0.60	0.80	0.68	n.a.	0.73	0.55	0.63
	MC-Dropout	VR	0.74	-0.37	0.76	0.96	0.85	-0.73	0.84	0.78	0.81	0.50	-0.69	0.56	0.87	0.68	-0.84	0.71	0.57	0.63
		PE	0.74	-0.12	0.76	0.96	0.85	-0.43	0.83	0.79	0.81	0.50	-0.48	0.61	0.78	0.69	-0.51	0.75	0.53	0.62
		MI	0.74	-0.39	0.75	0.98	0.85	-0.52	0.82	0.81	0.81	0.50	-0.52	0.52	0.93	0.67	-0.76	0.67	0.58	0.62
		MS	0.74	-0.07	0.77	0.96	0.85	-0.17	0.85	0.78	0.81	0.50	-0.40	0.61	0.79	0.69	-0.37	0.76	0.52	0.62

Table 2: Overview of supervision capabilities of different techniques, i.e., model types and quantifier combinations for different thresholds.

MS] (see Section 3). We want to investigate whether any combination of estimator type and quantifier dominates all the others in terms of S_1-score. To investigate how performance changes with the uncertainty threshold t, we consider different acceptable rates ϵ of false positives on the nominal data and we compute the threshold t that ensures such FPR on the validation set, so that we can compare alternative estimators/quantifiers at equal FPR on the validation set of the nominal data.

RQ_3 (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.

RQ_4 (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 each hyperparameter choice, so as to identify the regions where such standard deviation is low.

6.2 Subjects

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

Mnist [20] Classification of hand-written digits, formatted as small grayscale images. This is the most popular dataset in machine learning testing [31], and a relatively easy problem, where even simple models achieve high accuracy. We took the DNN architecture from a Keras tutorial [16].

Cifar10 [17] Classification of colored images into ten different classes. It is also very popular in DLS testing [31] and it represents a more challenging task than Mnist. We use the model architecture proposed in the Brownlees Cifar10 tutorial [4].

Traffic [34] Classification of images of European traffic signs [1,3,10,19,22,33,36,40]. 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 use was proposed alongside the release of this dataset [34].

Imagenet [6] (Pretrained) Image classification problem with as many as 1,000 classes. We use eight pre-trained *Efficientnet* models [38]. 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.

6.3 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 [24] as OOD test set for Mnist and the color-image transformations proposed by Hendrycks [11] 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 ϵ, with $\epsilon \in \{.01, .5, .1\}$ 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[6]. Due to the high workload, the training and prediction processes were distributed on three different workstations using Windows or Ubuntu and four different GPUs (one workstation had two GPUs). UNCERTAINTY-WIZARD was used for training, prediction and uncertainty quantification. [7]

6.4 Results

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

[6] Predictions were cached, such that for the evaluation of different sample sizes and different numbers of atomic models, previous predictions could be re-used.

[7] Replication package available at:
github.com/testingautomated-usi/repli-icst2021-uncertainty

	Technique	Per Subject mnist N=6	cifar10 N=6	traffic N=6	Overall Trained N=18	Pre-Trained N=48
Point Pred.	SM	3.33	9.67	**2.42**	5.14	3.56
	PCS	**2.67**	9.83	3.17	5.22	3.5
	SME	5.67	10.0	2.92	6.19	4.42
MC-Dropout	VR	5.83	5.83	5.67	5.78	3.55
	PE	7.0	6.67	5.5	6.39	4.5
	MI	6.67	6.83	5.75	6.42	5.12
	MS	5.33	7.17	6.58	6.36	**3.34**
Ensemble	VR	4.17	3.17	4.5	**3.94**	n.a.
	PE	8.17	**1.33**	10.25	6.58	n.a.
	MI	10.17	3.0	10.25	7.81	n.a.
	MS	7.0	2.5	9.0	6.17	n.a.

Table 3: Rank-Order Analysis: S_1-Score ranks, averaged over $\epsilon \in \{0.01, 0.05, 0.1\}$, nominal and OOD datasets

6.4.1 RQ1 (Effectiveness)

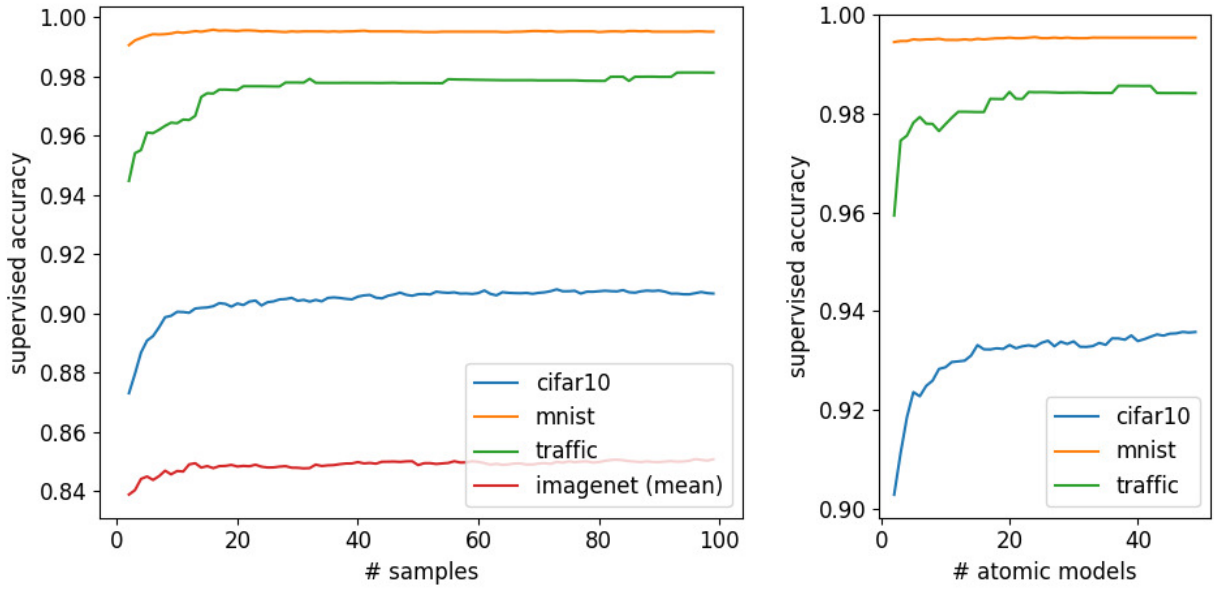
An overview of our results is provided in Table 2. Due to space constraints, the results for $\epsilon = 0.05$ are omitted in the table and the values for the 8 different Imagenet models are averaged. The full set of results can be found in the online replication package.

Our results suggest that all supervisors lead to supervised accuracies $\overline{ACC}$ which are at least as high, but typically much higher than the accuracy 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.*

6.4.2 RQ2 (Comparison)

If we look at the S_1-Score in Table 2, it is apparent that there is no uncertainty quantifier which outperforms all the other ones on every subject/dataset and for every threshold (ϵ). 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 3, 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 [29]), 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 theoretically well founded MC-Dropout. So, in practical cases as those considered in our experiments, Point Predictors may represent a good trade-off between performance and computational cost.



(a) (b)

Figure 2: Influence of sample size in MC-Dropout (a) and number of atomic models in Ensemble (b) on supervised accuracy $\overline{acc}$; values taken with $\epsilon = 0.1$ and MS quantifier on nominal dataset

Summary (RQ2): *There is no dominant supervisor, i.e., no supervisor which performs best for every test subject, data source and threshold. Ensembles are ranked generally well across subjects and thresholds, while Point Predictors offer a valuable trade off between performance and execution cost.*

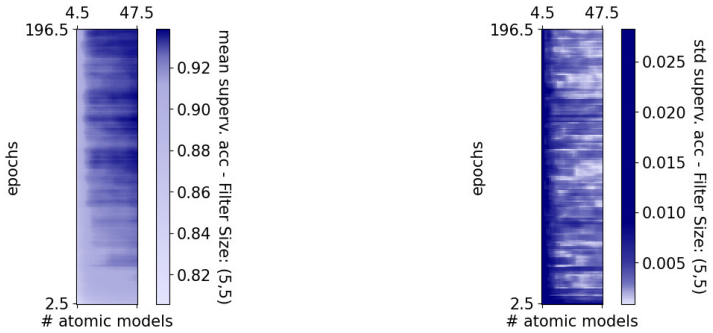
6.4.3 RQ3 (Sample size)

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 [29], 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 ϵ, because thresholds are correspondingly also discrete and limited to the number of samples. So, to achieve the target FPR ϵ 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).*

6.4.4 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 (*std*) of $\overline{ACC}$ within a 5x5 filter applied to this grid. In this way, we account for the variability of the supervised accuracy



(a) (b)

Figure 3: Average (a) and standard deviation (b) of the Ensemble's supervised accuracy $\overline{acc}$ with $\epsilon = 0.1$ on OOD data, computed for the Traffic subject inside a 5x5 neighbourhood of each configuration

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 $\overline{ACC}$ 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 $\overline{ACC}$: 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 2, 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.*

6.5 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 5 are specifically designed for such assessment.

6.6 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 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.

7 Related Work

Empirical Studies of Uncertainty-Aware Deep Neural Networks: Oliveira *et al.* [27] 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.* [29] 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 variational BNN approximations. Zhang *et al.* [44] 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.* [2] 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 can not help against aleatoric uncertainty. Stocco *et al.* [37] and Henriksson *et al.* [12] 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 predictions of the DNN being supervised.

8 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

[1] Rachid Belaroussi, Philippe Foucher, Jean-Philippe Tarel, Bahman Soheilian, Pierre Charbonnier, and Nicolas Paparoditis. Road sign detection in images: A case study. In *2010 20th International Conference on Pattern Recognition*, pages 484–488. IEEE, 2010.

[2] David Berend, Xiaofei Xie, Lei Ma, Lingjun Zhou, Yang Liu, Chi Xu, and Jianjun Zhao. Cats are not fish: Deep learning testing calls for out-of-distribution awareness. In *The 35th IEEE/ACM International Conference on Automated Software Engineering*, New York, NY, USA, 2020. Association for Computing Machinery.

[3] Igor Bonaci, Ivan Kusalic, Ivan Kovacek, Zoran Kalafatic, and S Segvic. Addressing false alarms and localization inaccuracy in traffic sign detection and recognition. In *16th computer vision winter workshop*, pages 1–8. Citeseer, 2011.

[4] Jason Brownlee. How to develop a cnn from scratch for cifar-10 photo classification, 2019.

[5] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning - ICML06*. ACM Press, 2006.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[7] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[8] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1050–1059. JMLR.org, 2016.

[9] GithubCompare. *Github statistics comparison between tensorflow and pytorch*, Accessed September 25, 2020.

[10] Cosmin Grigorescu and Nicolai Petkov. Distance sets for shape filters and shape recognition. *IEEE transactions on image processing*, 12(10):1274–1286, 2003.

[11] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *International Conference on Learning Representations*, 2018.

[12] Jens Henriksson, Christian Berger, Markus Borg, Lars Tornberg, Cristofer Englund, Sankar Raman Sathyamoorthy, and Stig Ursing. Towards structured evaluation of deep neural network supervisors. In *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, apr 2019.

[13] Eddy Ilg, Ozgun Cicek, Silvio Galesso, Aaron Klein, Osama Makansi, Frank Hutter, and Thomas Brox. Uncertainty estimates and multi-hypotheses networks for optical flow. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 652–667, 2018.

[14] Laurent Valentin Jospin, Wray Buntine, Farid Boussaid, Hamid Laga, and Mohammed Bennamoun. Hands-on bayesian neural networks – a tutorial for deep learning users, 2020.

[15] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5574–5584. Curran Associates, Inc., 2017.

[16] Keras-Team. Code: Convolutional neural network example, 2018.

[17] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009.

[18] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pages 6402–6413, 2017.

[19] Fredrik Larsson and Michael Felsberg. Using fourier descriptors and spatial models for traffic sign recognition. In *Scandinavian conference on image analysis*, pages 238–249. Springer, 2011.

[20] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[22] Markus Mathias, Radu Timofte, Rodrigo Benenson, and Luc Van Gool. Traffic sign recognition—how far are we from the solution? In *The 2013 international joint conference on Neural networks (IJCNN)*, pages 1–8. IEEE, 2013.

[23] Rhiannon Michelmore, Marta Kwiatkowska, and Yarin Gal. Evaluating uncertainty quantification in end-to-end autonomous driving control. *CoRR*, 2018.

[24] Norman Mu and Justin Gilmer. Mnist-c: A robustness benchmark for computer vision. *arXiv*, 2019.

[25] Radford M Neal. Bayesian training of backpropagation networks by the hybrid monte carlo method. Technical report, Citeseer, 1992.

[26] David A Nix and Andreas S Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 ieee international conference on neural networks (ICNN'94)*, volume 1, pages 55–60. IEEE, 1994.

[27] Ramon Oliveira, Pedro Tabacof, and Eduardo Valle. Known unknowns: Uncertainty quality in bayesian neural networks. *Workshop on Bayesian Deep Learning, NIPS 2016, Barcelona, Spain*, 2016.

[28] Ian Osband. Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *Neural Information Processing Systems (NIPS)*, 2016.

[29] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your models uncertainty? evaluating predictive uncertainty under dataset shift. *Advances in Neural Information Processing Systems*, pages 13991–14002, 2019.

[30] Tim Pearce, Felix Leibfried, and Alexandra Brintrup. Uncertainty in neural networks: Approximately bayesian ensembling. In *International conference on artificial intelligence and statistics*, pages 234–244. PMLR, 2020.

[31] Vincenzo Riccio, Gunel Jahangiroba, Andrea Stocco, Nargiz Humbatova, Michael Weiss, and Paolo Tonella. Testing machine learning based systems: a systematic mapping. *Empirical Software Engineering*, 2020.

[32] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, 10(3):e0118432, mar 2015.

[33] Siniša Šegvic, Karla Brkić, Zoran Kalafatić, Vladimir Stanisavljević, Marko Ševrović, Damir Budimir, and Ivan Dadić. A computer vision assisted geoinformation inventory for traffic infrastructure. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 66–73. IEEE, 2010.

[34] Citlalli Gámez Serna and Yassine Ruichek. Classification of traffic signs: The european dataset. *IEEE Access*, 2018.

[35] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[36] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32:323–332, 2012.

[37] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. Misbehaviour prediction for autonomous driving systems. In *Proceedings of 42nd International Conference on Software Engineering*, page 12 pages. ACM, 2020.

[38] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[39] Brad Templeton. Tesla in taiwan crashes directly into overturned truck, ignores pedestrian, with autopilot on, 2020.

[40] Radu Timofte, Karel Zimmermann, and Luc Van Gool. Multi-view traffic sign detection, recognition, and 3d localisation. *Machine vision and applications*, 25(3):633–647, 2014.

[41] C.J. van Rijsbergen. *Information Retrieval, 2nd Edition, Chapter 7*. Butterworths, 1979.

[42] James Vincent. Google 'fixed' its racist algorithm by removing gorillas from its image-labeling tech, 2018.

[43] Michael Weiss and Paolo Tonella. Uncertainty-wizard: Fast and user-friendly neural network uncertainty quantification. *arXiv preprint arXiv:2101.00982*, 2020.

[44] Xiyue Zhang, Xiaofei Xie, Lei Ma, Xiaoning Du, Qiang Hu, Yang Liu, Jianjun Zhao, and Meng Sun. Towards characterizing adversarial defects of deep learning software from the lens of uncertainty. In *Proceedings of 42nd International Conference on Software Engineering*. ACM, 2020.