Making the Relationship between Uncertainty Estimation and Safety Less Uncertain

Vincent Aravantinos
Autonomous Intelligent Driving GmbH
Munich, Germany
vincent.aravantinos@aid-driving.eu

Peter Schlicht
Volkswagen Group Research
Wolfsburg, Germany
peter.schlicht@volkswagen.de

***Abstract*—Uncertainty estimation is an intense subject of research in object detection. A commonly mentioned motivation is the usage of object detection in safety-critical applications. However, the precise connection between uncertainty estimation and safety engineering is seldom done: how shall uncertainty estimation really support safety engineering? How does uncertainty relate to classical safety engineering activities? Which problems does uncertainty estimation solve? Which new challenges does it pose? This paper sketches those connections more precisely and provides perspectives for future work in this direction.**

***Index Terms*—Deep Learning, Uncertainty, Safety**

I. Introduction

Usage of probabilities is prevalent in robotics [1]: it belongs to the fundamental tools of data fusion [2], it is essential for behavior prediction [3] and it propagates all along the chain of the classical sense-plan-act architecture [4]. It is therefore essential that any technology contributing to one of those blocks is able to generate not only a result, but a probability distribution of results. Obtaining such a distribution for deep neural networks (DNNs) is the subject of *uncertainty estimation*.

Uncertainty in deep learning is an intense subject of research [4]–[8]: *defining* uncertainty (ex. depending on the causes of this uncertainty: see aleatoric vs epistemic uncertainty), *assessing* uncertainty (ex. using Bayesian networks or Monte Carlo dropout), and assessing uncertainty *efficiently* have been and, to a certain extent, remain challenges.

As mentioned, estimating uncertainty is essential for probabilistic robotics thus making it essential to have DNNs generating such uncertainties. Another commonly mentioned motivation is the usage of uncertainty estimation for safety, especially in object detection to support safety-critical applications. *This paper focuses only on this usage of uncertainty estimation.*

The main objective of this work is to make more explicit the connection between uncertainty estimation and safety engineering. Indeed, we argue that this connection is still unclear as of the moment of writing. According to [4], "epistemic uncertainty is required to understand examples which are different from training data". But nothing more is mentioned. According to [7]: "Current state-of-the-art object detectors cannot determine whether a prediction is a true or false positive. This is undesirable for safety-critical systems like autonomous driving in which a wrong detection can have fatal consequences." Again, there are no more explanations about the matter. In [8], the authors are even more assertive: "Quantifying the uncertainty associated with the prediction of neural networks is a prerequisite for their deployment and use in safety-critical applications."

In summary, despite a clear "gut feeling" that uncertainty can help in the development of safety-critical systems, it is is still very vague how uncertainty relates to safety. How does it connect to classical safety engineering notions like failure modes? How does it connect to safety diagnostic and mechanisms? What are the parallels between uncertainty and the content of current standards? It is of course expected that no current standard will prescribes how to develop deep neural networks, however, there must be some parallels or similarities in rationales that can for sure be identified.

In this paper, we attempt to sketch more concretely those connections and parallels. We first introduce preliminaries of both worlds: in Section II, we define uncertainty and present current techniques to estimate this uncertainty; in Section III, we introduce basics of safety engineering, as relevant for the comparison. Section IV addresses combining both worlds by establishing parallels and connections between them. Section V highlights challenges that those parallels highlight and which we identify as avenues for future work: what is still missing to ensure that uncertainty estimation indeed fulfills the promises for the development of safety critical systems?

II. Uncertainty

The progress in the field of Deep Learning has brought Deep artificial neural network seem to be the tool of choice for realizing perception also for safety-critical applications such as autonomous driving. Those systems have to act in an open world and coexist and cooperate with humans. One challenge in this respect is the limited information, those perception systems can access. This has three major reasons:

1) intrinsic uncertainty in the world (random effects, non-predictable influences)
2) the limited sensing of autonomous systems even if we allow car-to-car communication (e.g. occlusion)
3) uncontrollable and hard to model behavior of others (e.g. pedestrians in traffic)

This limitation in information forcibly means that autonomous systems have to deal with uncertainty.

978-3-9819263-4-7/DATE20/©2020 EDAA

What adds to this is the imperfection of the DNNs themselves: besides the given uncertainty in the environment (the *aleatoric uncertainty*), perception systems do have model-specific inaccuracies leading to what is called the *epistemic uncertainty*.

As humans do as well, we would expect an autonomous agent to act more defensively or doubt its perception in situations dominated by measurably higher uncertainty in order to raise the safety of its actions. This gives strong motivation into estimating the amount of uncertainty in perception DNNs, which is usually done by measuring or propagating dispersion metrics on the DNN output interpreted as an estimated probability distribution over the possible classes.

For the sake of simplicity, we will stick to the case of a classification DNNs in this section, but the arguments generalize well to dense classification tasks like semantic segmentation, regression tasks or combinations thereof (like object detection).

Classification models usually output *Softmax* vectors wherein the class specific neuron activation values (*logits*) a_c for classes $c = 1, .., n$ are transformed to a class probability vector $(v_c)_{c=1,..n}$ where $v_c = \frac{\exp(a_c)}{\sum_{c=1,..,n} \exp(a_c)}$.

Interpreting this as a probability distribution over the possible classes allows for first uncertainty estimates $u = u((v_c)_{c=1,..n})$:

- most naively, the probability of the winning class yields an uncertainty estimate (i.e. $u = 1 - \max((v_c)_{c=1,..n})$)
- more statistically rigorous uncertainty estimates come from the variance $u = \mathrm{var}((v_c)_{c=1,..n})$ or the standard deviation $= \mathrm{std}((v_c)_{c=1,..n})$.
- other uncertainty estimates stem from measuring the difference between most and second most probable class of the output: $u = \max((v_c)_{c=1,..n}) - \max(\{v_c, c = 1, ..n\} \setminus \{\max((v_c)_{c=1,..n})\})$
- those measures can be combined to improve the quality of the estimated uncertainty [9]

The above estimates tend to suffer from the fact that classification DNNs - even though being interpretable as probabilistic models - are not trained towards outputting realistic probability vectors: usually, we provide the *Ground Truth* in form of *one-hot encoded vectors*, i.e. as Dirac Delta distributions that put all probability to the correct class. As we minimize the class specific deviation from the Ground Truth in supervised training, we motivate DNNs into reporting their output *over-confidently*. This effect can mitigated through two different approaches:

1) Network calibration - see [10] for an overview. One quite direct calibration approach is rescaling of the logits. This is sometimes referred to as *temperature scaling* and introduces another hyperparameter into the model development process. It can either be learnt or fine-tuned on a particular tuning set. Experiments show some improvement of the resulting uncertainty estimate - measured for example by means of quantifying the performance of the uncertainty estimate as a detector of wrong classifications. One possible metric for this is the *Area under Curve* for the *Receiver Operator curve* or the *Precision Recall curve*.
2) An alternative to calibration of DNNs lies with sampling approaches. Within those, some noise signal is applied either to the input or to intermediate activations within the network at inference. Inferring the network multiple times while sampling from those noise sources allows for better uncertainty estimates stemming from the averaged Softmax outputs: Droupout layers randomly switch off connections within a DNN. Gal has shown that sampling from a DNN with active dropout layers yields more realistic uncertainty estimates [5]. Alternatively, there are approaches of a latent probability space being learned alongside the training process. This latent space is expected to encode the experienced uncertainty through adding noise to particular layers close to the model output. [11]

A downside of the sampling approaches for measuring the DNN uncertainty is their impact on the need for computing power and the increased latency on embedded devices. Postels et al propose some possible solution by introducing a sample-free method of uncertainty estimation which essentially propagates the noise coming from Dropout layers in an efficient way through the remaining network [8].

III. Safety Engineering

Safety engineering is a long-standing discipline described both in academic literature and in numerous standards and norms. In this paper, we will mainly refer to automotive standards, especially the ISO26262 [12] (emerging standards like SOTIF [13] might also be considered) but most concepts should apply equally to other approaches to safety engineering as used in other industries, ex. railway or aeronautic [14].

A. System level

A first point to make, which characterizes a frequent misconception in the ML community: safety engineering is, before all, a *system engineering* discipline. A system is typically an entire car, plane or train. This is to oppose to a neural network, which is only a very small subpart of the entire system. As a consequence, it is impossible to devise techniques "to make a neural network safe": the entire system is safe or not, but not a single part of it.

From a methodological perspective, this means that safety engineering typically starts by considering hazards at the level of the entire system, then deriving safety goals from a "HARA" (Hazard Analysis and Risk Assessment). Safety goals are basically safety-specific system-level requirements, associated with some so-called integrity level defining how much rigour shall be put in the fulfilment of those requirements: in the ISO-26262, such integrity levels are graded from A (lowest level) to D (highest level). From safety goals, one derives a safety concept (essentially an architecture of the system guaranteeing that the safety goals are reached), which is itself made more and more concrete until we reach the level of designing

particular pieces of software and hardware with dedicated refined safety requirements. *Only then do we encounter the possibility of having to consider safety requirements for a neural network.* Since this paper focuses however on neural networks, we will not cover all those early phases but it is essential to be aware of them to understand what can be solved or not at the level of a neural network.

This paper therefore positions itself at the same level of safety standards usually applying at the level of components, ex. software or hardware. In the ISO-26262, this corresponds to parts 5 (hardware) and 6 (software). In aeronautics, this corresponds to the DO-178C [14] for software.

B. Safety measures and safety mechanisms

Once we reach the component level, we have safety requirements imposing both which failures are safety-critical *and* which integrity level they shall have. There are then different (complementary) ways to reach those integrity levels. Some such ways are to apply at design time and some others at run-time. Typical examples at design time ("safety measures") include using safety-related design patterns, ex. redundancy, which enables to reduce failure rates on condition of independence. Other examples are purely process-related, ex. for software: "statement coverage is highly recommended to assess the coverage of tests for ASIL-A" or "MC/DC (Modified Condition/Decision Coverage is highly recommended to assess the coverage of tests for ASIL-D", etc.

Design-time methods typically have the objective of reducing (or even completely eliminating) failures before the system is deployed. Runtime methods, on the other hand, target failures *once* the system is deployed. Typically, if no safety measure can provably reduce the risk of failure sufficiently that the desired integrity level is reached, then one needs to find alternative ways ("safety mechanisms") to address the failures. In such a case, the objective is not to prevent the failure from happening, but rather to 1. detect it and 2. react to it. The failure rate per se is not diminished but one can define mechanisms ensuring that the (risky) consequences of the failure are being circumvented.

Defining such mechanisms requires to:

1) list out all the fault models (i.e., types of faults) that can entail the unwanted failure,
2) for all those fault models, design diagnostic mechanisms enabling to detect whether a fault occurred or not,
3) for every potential fault occurrence, design reaction mechanisms to ensure that, either the fault does not result into a failure, or the failure is clearly communicated to the handling components at a higher level.

Various fault diagnostic mechanisms can be found, particularly for hardware, in the appendix D of the ISO26262:5 [12].

C. Systematic and random failures

Note that diagnostic mechanisms are typically encountered to mitigate so-called *random* failures. Random failures are typically encountered for hardware: the same hardware tested twice in the same situation can randomly generate a different outcome. Of course both situations are never exactly the same, but the extent of our knowledge makes both situations non distinguishable (ex. we cannot know the state of every atom in the world at every instance in order to compare both situations). Software failures on the other hand are typically so-called *systematic* failures: the same software – at least at an algorithmic level – shall return the same outcome given the same input and the same internal state (and this state shall be in control of the engineer – which is not the case for hardware).

An ideal way of assessing whether an integrity level is reached is to estimate the frequency of occurrence of a failure. This is typically doable for random failures but very hard for systematic ones. Indeed, failure rates of random failures are (very approximately) obtained by letting the system run a long time and simply measuring how often it fails. Even though it is in principle doable for systematic failures, it does not make much sense: when a systematic failure is encountered, it is reproducible and there is zero acceptable reason for not fixing it. It is therefore very hard to obtain some reasonable statistical estimation of a failure rate for systematic failures and, so far, those have been tackled essentially by process-based measures: in the absence of a simple way to actually estimate the failure rate, standards typically skip this estimation and just define a set of processes to follow depending on the ASIL, ex., as mentioned above, for software: "statement coverage is highly recommended to assess the coverage of tests for ASIL-A".

IV. Combining both worlds

We now assume that the neural network is used to fulfill a safety-critical requirement (say, some ASIL-D requirements). This section aims at mapping uncertainty estimation to parts of safety engineering where it makes the most sense.

Clearly, in the methods highlighted in the previous section, the closest form of safety measure/mechanism to which uncertainty estimation can relate is runtime safety mechanisms: just like for, say, hardware random faults, uncertainty estimation can be considered as diagnostic method to identify, at runtime, when a particular fault might trigger. It can also be used to arbitrate between different, possibly contradicting, inputs in the form of a voter mechanism.

Let us now investigate this analogy. As mentioned earlier, to define such a safety mechanism, one must: 1. list out all potential fault models, 2. design detection mechanisms, 3. react to fault detections.

A. Fault models for DNNs

A first step is to analyze the DNN and list out potential fault models. Note that once the industry becomes more mature, it will be possible to sketch typical fault models in a standard, as is done for instance for hardware in the ISO26262. For now, it is still unclear whether safety concepts for autonomous driving or ADAS need to rely on the functions fulfilled by DNNs, and if so how (is a false positive safety critical? is confusing a car for a tree safety critical? etc.). Furthermore, those functions are even not yet stable across industries (ex. 2D image-based detectors vs 3D Lidar-based

ones). As a consequence, it is unlikely to expect, like for hardware, that a typical list of fault models for DNNs appear in the standard anytime soon; therefore, any development of DNNs for safety-critical applications nowadays shall include an activity explicitly investigating the fault models of the DNNs.

Note that fault models depend actually only on the *function* that a DNN has to fulfill (ex. "detect pedestrians in a Lidar point cloud") and not on the fact that we use DNNs or not. As a consequence, even if rule-based algorithms were used to fulfill the function, the same faults could occur and would be relevant (but one would handle them typically differently: by inspecting the rules of the algorithm to understand how the fault might occur).

Example 1: Assume that a DNN is used as a 3D object detector taking as input Lidar and camera data and returning 3D bounding boxes (x, y, yaw, w, h, l) paired with a classification (pedestrian, bicycle, car).

Typical faults considered in the ML literature are: false negative, false positive, bad classification, wrong yaw, wrong size, wrong position. In practice, the fault models shall depend on the safety requirements. Even though this paper assumes that the safety requirements are done separately and therefore makes it impossible to actually define the fault models, experience makes it clear that a fault model as simple as "false negative" will not be enough. Typically, safety requirements include tolerance margins and would distinguish different faults depending on their impact. Therefore, a more realistic list of fault models could be:

- instead of just "false negative": false negative *for a pedestrian*, false negative *for a bicycle*, etc.
- idem for false positives,
- instead of just "bad classification": "car mistaken for a pedestrian","pedestrian mistaken for a car", etc.
- instead of just "wrong yaw": "yaw wrong by less than 5°", "yaw wrong by less than 10°" (those are not exclusive: one might want to consider both failure modes but with different integrity levels), etc.
- idem for position and size.

It shall be clear from these examples that, in order to be useful for safety, measuring the performance of a DNN goes way beyond a simple F1-score.

A first lesson to learn based on these examples is that, from a safety perspective, obtaining a *continuous* uncertainty is not relevant: faults are boolean, either we have a fault or we do not. The continuous value is then only a tool to assess the presence/absence of the fault.

B. Uncertainty as a detection mechanism

At that stage, the connection to uncertainty is obvious: uncertainty estimation is a safety mechanism that (potentially) allows to detect the occurrence of a given fault. As a consequence, a DNN estimating uncertainty shall do so according to the fault models defined in earlier phases.

From a safety engineering perspective, there is still a lot of work to do to obtain a proper detection mechanism thought:

1) How to react to the fault itself: this is typically some architectural work, which has to be done independently of the ML engineers.
2) It shall be assessed how much the detection mechanism detects faults: How often does it *indeed* detect faults? How many of these faults still go undetected? This is called the "diagnostic coverage" in the ISO26262.
3) Once this diagnostic coverage is assessed, one can analyze whether the integrity level is reached or whether further measures shall be taken.

Points 1 and 3 are independent of the detection mechanism itself and are therefore not of interest for uncertainty estimation and for this paper. Points 2 however rise challenges for uncertainty estimation that might be of interest for the research community. We explore those in Section V.

C. Usage of uncertainty in the reaction mechanism

Once a fault is detected (using uncertainty or not), one needs to react to the fault to mitigate it. A particular form of fault can involve different components providing contradicting outcomes. In such a case, one typically needs a *voter* to arbitrate between both results. Even if uncertainty is not used for the fault detection, it could be used as a mechanism to mitigate the fault. In this paper, we focus on fault detection. The usage of uncertainty for voter mechanisms is however to be explored further.

D. Note on the pseudo-randomness of DNNs' failures

As mentioned earlier, runtime safety mechanisms are mostly encountered for random failures. DNNs are however software and shall therefore only be subject to systematic failures. How can we then understand that uncertainty is then used in a way that is usually employed for random failures more than for systematic ones? We explore this question in that section.

Classical connections between machine learning and safety engineering is to consider DNNs as software and thus to tackle essentially the sources of systematic failures in a similar manner as is classically done for software. Various recent works acknowledge this and attempt therefore to transpose some techniques from software engineering to neural networks, see ex. [15]–[18].

We argue that deep neural networks, despite being software, also exhibit a *form* of random failure. Indeed, given two very similar inputs, ex. an image with a pedestrian and the same image in which the same pedestrian is slightly shifted, a deep neural network might generate two different results. Of course, this does not literally exhibit a random failure: both inputs are different, so there is in principle no reason to assume that the outputs shall be the same, from a pure algorithmic point of view. However, as explained earlier, we can argue that the same holds for random failures in hardware: even if a piece of hardware fails for what seems to be the exact same situation as when it did not fail, the reality was actually not exactly the same when executed (not all atoms were exactly in the same state).

The main similarity however is that, in both cases, the differences between how the system reacts to two very similar, yet different, situations is not completely in our hands: we do not *know* how a piece of hardware will react to a small change in the atoms of a given situation, just as we do not *know* how a (esp. deep) neural network will react to two very similar but yet different images (at least, as long as explainable AI remains a non-solved endeavor). This is very different for classical "rule-based" algorithms, for which we know the logic processing some inputs.

This situation can be summarized by the following quote from [19]:

> If our knowledge of an event that has already happened is incomplete then we need to reason about it in the same way as we reason about uncertain events yet to happen.

Note: this analogy is the reason why talked so far of "fault model" for DNNs instead of "failure mode". This choice might be of course disputable.

V. Challenges

If one wants to use uncertainty estimation for safety, it is essential to guarantee that this estimation indeed does what it is intended to do: provide an estimation of how "wrong" the network can be. More precisely, we need a way to assess how realistic the uncertainty estimation indeed is. There are two different levels to address this: one at the uncertainty estimation level, the other one at the fault detection level.

A. Uncertainty realism

Picture the following situations: imagine we use two different 3D object classifiers. Both classifiers provide some uncertainty estimation over the classification in the form of a real number between 0 and 1. Suppose we provide as input a point cloud in which a pedestrian is present. Both classifiers identify the pedestrian but they assign different uncertainties. How can we decide which one provides the "best" uncertainty? Which of both uncertainties is the most reliable?

There are various directions to tackle this.

1) Uncertainty "realism": Various approaches attempt at assessing how realistic uncertainty is [20], [21]. Those approaches are interesting but very hard since no ground truth is available.

2) Statistics: A first possibility is to take inspiration from statistics: this is not the first time in History that one tries to estimate some probability distribution. This has been the subject of statistics for a long time.

In statistics, the typical way of obtaining such a probability distribution is to rerun the "experiment" in order to obtain new samples, for which we would get new outcomes because experiments are non-deterministic (at least those considered in the realm of statistics). Out of the multiple outcomes, one could then infer a probability distribution (typically using classical statistical tests). Here again, the question of selecting, among various possible models, which one is the best, is a very classical problem in statistics, called "model selection" [22]. Typically a model is considered better than another if it fits better the data and if it is simple enough (a form of Occam's razor to avoid overfitting), see for instance the Bayesian Information Criterion (BIC [23]).

In the case of DNNs, it is however quite different: rerunning the DNN in the same situation shall always return the same result, so that we do not actually obtain a distribution. Current approaches to uncertainty estimation attempt to reproduce a similar situation by sampling different DNNs. But, contrarily to experiments just mentioned, such experiments do not provide any guarantee since the various sampled outcomes are provided by the DNNs themselves. *In conclusion, it is actually very hard to assess what the uncertainty provided by DNNs actually represents.*

Parallels with classical statistics remain to be explored.

3) Uncertainty labelling: A pragmatic possibility is to try to label uncertainty, as much as possible. For instance, one could request labelled data where not only a bounding box is provided, but a bounding box paired with some intervals around the borders. Such intervals could represent both the uncertainty of the "labeler" or uncertainty due to sensor limitation (in which case the uncertainty can be automatically assigned, ex. camera resolution). This would not handle epistemic uncertainty but would provide maybe more ground to aleatoric uncertainty.

Whether used during training or not (not all uncertainty estimation methods are able to take this form of training data into account), having such a labelled data could at least provide a test set allowing for proper validation of the uncertainty.

4) Fault detection: All the previous directions attempt at validating the uncertainty estimations themselves, i.e., some real number between 0 and 1 in the easiest case, or some probability distribution in the most complex case. However, if the purpose of uncertainty estimation is only to detect faults, then we only want to know if the estimation passed a particular threshold. As a consequence, the best way to make sure that the uncertainty estimation is usable for safety might not be trying to validate the uncertainty itself but rather to measure how often does this uncertainty enables to find a fault. This would simplify the task a lot and be sufficient to answer the needs from a safety engineering perspective. It would allow to get rid of questions like "what is the meaning of a given uncertainty number in reality?".

If such a direction was to be pursued, the overall assessment would be much simpler: one would need to define appropriate thresholds *based on the fault models* as described in Section IV-A and run tests on a test set in order to obtain a diagnostic coverage for the given method. It is then enough to identify how many faults were detected and how many were not.

We consider this direction to be actually the most promising.

VI. Conclusion

In this paper, we tried to make more precise how and why uncertainty estimation can be useful for safety. To do so, we recalled both uncertainty estimation and the relevant methods from safety engineering. We presented analogies

between uncertainty estimation and safety mechanisms. We highlighted in particular the similarity with random hardware failures and took therefore inspiration of the hardware part of the ISO26262 (part 5, [12]) to sketch how to use uncertainty estimation for safety engineering, in particular to detect faults. This analogy conducted us to identify follow-up challenges in this area: it is not enough to detect a fault, one should also estimate the diagnostic coverage of this detection mechanism. This opens new follow-up research directions for uncertainty estimation and safety engineering.

REFERENCES

[1] S. Thrun, "Probabilistic robotics," *Commun. ACM*, vol. 45, no. 3, pp. 52–57, Mar. 2002. [Online]. Available: http://doi.acm.org/10.1145/504729.504754

[2] D. L. Hall and J. Llinas, "An introduction to multisensor data fusion," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 6–23, Jan 1997.

[3] D. Ridel, E. Rehder, M. Lauer, C. Stiller, and D. Wolf, "A literature review on the prediction of pedestrian behavior in urban scenarios," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2018, pp. 3105–3112.

[4] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5574–5584. [Online]. Available: http://papers.nips.cc/paper/7141-what-uncertainties-do-we-need-in-bayesian-deep-learning-for-computer-vision.pdf

[5] Y. Gal, "Uncertainty in deep learning," Ph.D. dissertation, University of Cambridge, 2016.

[6] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *International conference on machine learning*, 2016, pp. 1050–1059.

[7] M. T. Le, F. Diehl, T. Brunner, and A. Knol, "Uncertainty estimation for deep neural object detectors in safety-critical applications," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 3873–3878.

[8] J. Postels, F. Ferroni, C. Huseyin, N. Navab, and F. Tombari, "Sampling-free epistemic uncertainty estimation using approximated variance propagation," *submitted to International Conference on Computer Vision*, 2019.

[9] M. Rottmann, P. Colling, T. Hack, F. Hüger, P. Schlicht, and H. Gottschalk, "Prediction error meta classification in semantic segmentation: Detection via aggregated dispersion measures of softmax probabilities," *CoRR*, vol. abs/1811.00648, 2018. [Online]. Available: http://arxiv.org/abs/1811.00648

[10] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1321–1330.

[11] S. Kohl, B. Romera-Paredes, C. Meyer, J. De Fauw, J. R. Lesam, K. Maier-Hein, S. A. Eslami, D. J. Rezende, and O. Ronneberger, "A probabilistic u-net for segmentation of ambiguous images," in *Advances in Neural Information Processing Systems*, 2018, pp. 6965–6975.

[12] "Iso 26262: Road vehicles – Functional safety," 2011.

[13] "Iso/pas 21448: Road vehicles – Safety of the Intended Functionality," 2019.

[14] "Rtca do-178c: Software considerations in airborne systems and equipment certification," 2012.

[15] V. Aravantinos and F. Diehl, "Traceability of deep neural networks," *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1812.06744

[16] C.-H. Cheng, F. Diehl, Y. Hamza, G. Hinz, G. Nührenberg, M. Rickert, H. Ruess, and M. Troung-Le, "Neural networks for safety-critical applications - challenges, experiments and perspectives," 2017.

[17] R. Salay, R. Queiroz, and K. Czarnecki, "An analysis of iso 26262: Using machine learning safely in automotive software," 2017.

[18] A. Senyard, E. Kazmierczak, and L. Sterling, "Software engineering methods for neural networks," in *Tenth Asia-Pacific Software Engineering Conference, 2003.*, Dec 2003, pp. 468–477.

[19] N. Fenton and M. Neil, *Risk Assessment and Decision Analysis with Bayesian Networks*, 1st ed. Boca Raton, FL, USA: CRC Press, Inc., 2012.

[20] J. Sicking, A. Kister, M. Fahrland, S. Eickeler, F. Hüger, S. Rüping, P. Schlicht, and T. Wirtz, "A systematic approach for the assessment of neural network uncertainties," to appear.

[21] J. T. Horwood, J. M. Aristoff, N. Singh, A. B. Poore, and M. D. Hejduk, "Beyond covariance realism: a new metric for uncertainty realism," in *Signal and Data Processing of Small Targets*, I. S. for Optics and Photonics, Eds., 2014.

[22] P. Lahiri, *Model selection*, I. of Mathematical Statistics, Ed., 2001.

[23] H. S. Bhat and N. Kumar, "On the derivation of the bayesian information criterion," 2010.