Detection of outliers in classification by using quantified uncertainty in neural networks

Magnus Malmström, Isaac Skog, Daniel Axehill and Fredrik Gustafsson

Malmström, M., Skog, I., Axehill, D., Gustafsson, F., (2022), Detection of outliers in classification by using quantified uncertainty in neural networks, *25th International Conference of Information Fusion*. https://doi.org/10.23919/FUSION49751.2022.9841376

Original publication available at:
https://doi.org/10.23919/FUSION49751.2022.9841376

Detection of outliers in classification by using quantified uncertainty in neural networks

Magnus Malmström
Linköping University
magnus.malmstrom@liu.se

Isaac Skog
Linköping University
isaac.skog@liu.se

Daniel Axehill
Linköping University
daniel.axehill@liu.se

Fredrik Gustafsson
Linköping University
fredrik.gustafsson@liu.se

***Abstract*—Neural Networks (NNs) can solve very hard classification and estimation tasks but are less well suited to solve complex sensor fusion challenges, such as end-to-end control of autonomous vehicles. Nevertheless, NN can still be a powerful tool for particular sub-problems in sensor fusion. This would require a reliable and quantifiable measure of the stochastic uncertainty in the predictions that can be compared to classical sensor measurements. However, current NN's output some figure of merit, that is only a relative model fit and not a stochastic uncertainty. We propose to embed the NN's in a proper stochastic system identification framework. In the training phase, the stochastic uncertainty of the parameters in the (last layers of the) NN is quantified. We show that this can be done recursively with very few extra computations. In the classification phase, Monte-Carlo (MC) samples are used to generate a set of classifier outputs. From this set, a distribution of the classifier output is obtained, which represents a proper description of the stochastic uncertainty of the predictions. We also show how to use the calculated uncertainty for outlier detection by including an artificial outlier class. In this way, the NN fits a sensor fusion framework much better. We evaluate the approach on images of handwritten digits. The proposed method is shown to be on par with MC dropout, while having lower computational complexity, and the outlier detection almost completely eliminates false classifications.**

I. Introduction

This paper considers the problem of quantifying uncertainty in the output of neural networks (NNs), and how the knowledge about the uncertainty can be used to detect outliers. In applications such as computer vision [1], reconstruction of images [2], and various control tasks, e.g., reference tracking [3] and representation of the controller used for model predictive control [4], NNs have shown great prediction performance. Despite this success, there is a limited use of them in safety-critical applications [5–7]. One major reason for this is that quantification of the uncertainty in the output of NNs is still an immature area. Hence, in recent years it has received an increased attention, see e.g., [8–12]. For safety-critical applications, access to the variance (uncertainty) of the output of the model is crucial to know when it is necessary for a human operator to intervene in the decision-making process. Autonomous driving is an example of such a safety-critical application where it would be beneficial to include NNs in the decision-making process. For example, there is a need to classify images of objects in traffic scenarios, e.g., detecting pedestrians and interpreting speed signs.

As a particular example, the infamous Uber accident can be mentioned [13]. Uber decided to solely rely on the NN interpreting camera images of the environment, and they decided to disable the radar and laser scanner in the Volvo XC90 used in their test fleet. The NN was not trained to detect a pedestrian walking with a bike, and thus the NN failed completely to detect the pedestrian crossing the road. Hence the automatic braking system was not activated until the very last moment when it was too late and the pedestrian was killed. It is almost certain, but cannot be proven, that both the radar and laser scanner would have detected the pedestrian well in time to avoid the fatal accident. The example highlights the need to compute a reliability measure of the classifier output.

The goal of this paper is to embed the NN in a stochastic framework enabling a method to propagate a stochastic uncertainty from the training phase to a probabilistic classifier output. As a result, the output from the NN based sensor can be weighed together with information from multiple sources in a sensor fusion framework. In the Uber case, the obstacles detected by the NN based vision system can be merged with radar and laser scanner information.

Monte Carlo (MC) dropout is a state-of-the-art method for quantifying uncertainty in NN [14]. In MC dropout, parameters of the NN are randomly set to zero, and in that way, an ensemble of models is obtained. Each model gives one classifier output, which can be seen as an MC sample from the classifier. The stochasticity in the method comes from how the parameters are set to zero. This paper proposes an alternative method to quantify the uncertainty in the prediction of NN.

For the proposed method, in the training phase an approximative covariance of the parameters is computed. That is a local approximation that relies on the linearization of the NN. Since NN's typically has millions of parameters, one would face memory problems in storing the covariance matrix of all parameters. In this paper, we make the assumption, motivated by the success of transfer-learning, that it is the last layers that contribute the most to the uncertainty, while the first layers can be seen as feature extractors.

Linearization is a standard approach in system identification, see e.g., [15–21]. Here, linearization is primarily used for regression problems with least squares loss functions. In this paper, the linearization method is adapted to classification

This work is supported by Sweden's innovation agency, Vinnova, through project iQDeep (project number 2018-02700).

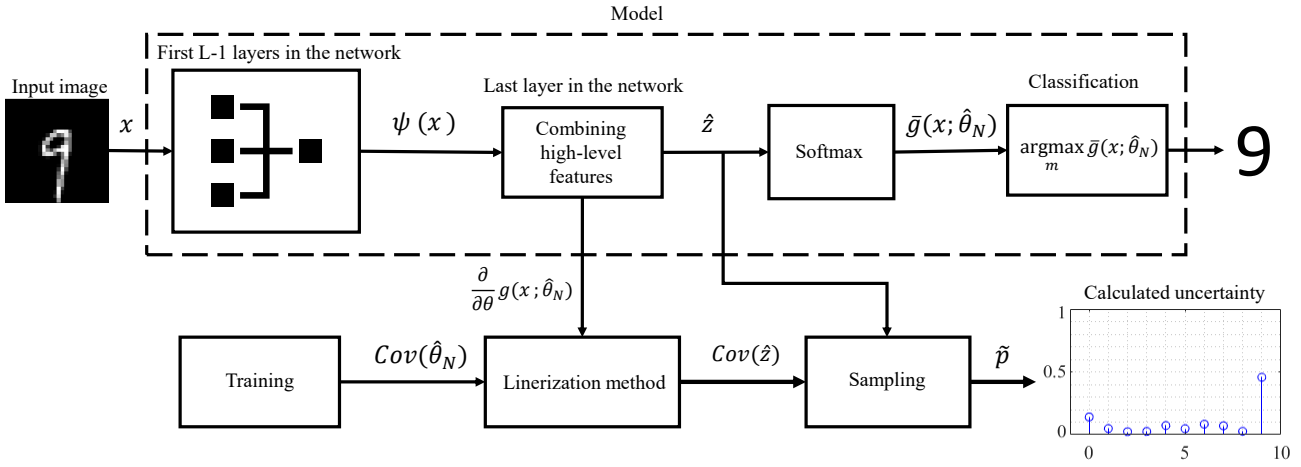


Fig. 1: Flowchart of the proposed method to quantify uncertainty in output of the classifier. The top part is the commonly used set-up for classification, while the lower part is the proposed extension used to quantify the uncertainty in the predictions, a.k.a. classifications.

problems with a cross-entropy loss function. Linearization of the NN's can be motivated by the large amount of data leading to a small uncertainty in the parameter estimate. Likewise, a large amount of data can motivate a Gaussian distribution of the parameter estimate based on the central limit theorem.

In the classification phase, using an assumption of a Gaussian distribution approximation, MC samples of the output of the NN are generated. Since the number of classes is usually rather small, the number of MC samples can be quite small, typically in the order of 1000. The evaluation of the last layers is very quick, so the computational burden in this approach is manageable. The flowchart for the proposed method to quantify uncertainty in the classification can be seen in Fig. 1.

The proposed method is compared to, the commonly used MC dropout method [14]. It is shown that they give similar results on the task of classifying handwritten digits in terms of the quantification of the uncertainty in the prediction. This while having lower computational complexity. To make the method more viable in practice, another contribution is to provide a recursive method to update the parameter covariance.

One additional use of the method is for outlier detection. We define an extra artificial outlier class, where inputs to the NN which do not lead to a reliable classification end up. Images in this outlier class can be handled separately in a later stage. For other methods to quantify uncertainty similar ideas have been presented see e.g., [22–24]. Compared to previous work, this paper emphasized that not many images have to be removed to get an almost perfect classification accuracy.

II. Neural networks used for classifications

Even though the method to quantify uncertainty in the classification proposed is valid for any parametric model, the focus is towards NNs. As they have proven superior in the task of classifying images [1, 25].

A. Problem formulation

An input $\mathbf{x} \in \mathbb{R}^{n_x}$ of size n_x is observed and the goal in the classification is to map the input to a class label $\beta \in 1, 2, \ldots, M$, which represents the M different classes. The general classification approach is to fit a model structure $\mathbf{g}(\mathbf{x}, \boldsymbol{\theta}) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_{\boldsymbol{\theta}}} \to \mathbb{R}^M$ with $n_{\boldsymbol{\theta}}$ parameters $\boldsymbol{\theta} \in \mathbb{R}^{n_{\boldsymbol{\theta}}}$ to training data $\{\mathbf{x}_n\}_{n=1}^N$ with corresponding labels $\{\beta_n\}_{n=1}^N$. After the training phase, the classifier can be expressed as

$$\hat{\beta}(\hat{\boldsymbol{\theta}}_N) = \arg\max_m g_m(\mathbf{x}; \hat{\boldsymbol{\theta}}_N), \tag{1}$$

where $\hat{\boldsymbol{\theta}}_N \in \mathbb{R}^{n_{\boldsymbol{\theta}}}$ are parameters found such that

$$\hat{\boldsymbol{\theta}}_N = \arg\min_{\boldsymbol{\theta}} \sum_{n=1}^{N} d_\beta(\beta_n, \hat{\beta}_n(\boldsymbol{\theta})), \tag{2a}$$

$$\hat{\beta}_n(\boldsymbol{\theta}) = \arg\max_m g_m(\mathbf{x}_n; \boldsymbol{\theta}). \tag{2b}$$

Here index m denotes the m:th value in the vector which corresponds to the m:th class. Basically, to describe the whole approach, the model structure, the chosen metric $d_\beta()$ to measure the distance between the true and predicted class, and the method to find the parameters $\hat{\boldsymbol{\theta}}_N$ which minimize this metric has to be decided. In the sequel, we will simply write $\hat{\beta}$ for the classifier output.

In practice, it is difficult to construct such a metric $d_\beta()$. Instead of training a model to estimate a given class β as in (2), it is a standard approach to train a model to estimate

$$\mathbf{y}_n = \begin{bmatrix} y_{1n} & \ldots & y_{Mn} \end{bmatrix}^\top \in \mathbb{I}^M, \tag{3}$$

that is a vector where element in $\mathbf{y}_n$ are the probability for the input to be annotated to any of the M classes. Here $\mathbb{I}^M$ denotes M-dimension a vector where all elements are between zero and one . An example of annotation is where $\mathbf{y}_n = \mathbf{e}_{\beta_n}$, where $\mathbf{e}_{\beta_n}$ denotes the unit vector with a one at position β_n. This is also known as a one-hot encoding. Then the parameters $\hat{\boldsymbol{\theta}}_N$ can be found such that

$$\hat{\boldsymbol{\theta}}_N = \arg\min_{\boldsymbol{\theta}} \sum_{n=1}^{N} d_y(\mathbf{y}_n, \bar{\mathbf{g}}(\mathbf{x}; \hat{\boldsymbol{\theta}}_N)). \tag{4}$$

To simplify the construction of the metric $d_y()$, a model $\bar{\mathbf{g}}(\mathbf{x};\boldsymbol{\theta}) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_{\boldsymbol{\theta}}} \to \mathbb{I}^M$ is commonly used. This can be done using the softmax function

$$\bar{\mathbf{g}}(\mathbf{x};\boldsymbol{\theta}) = \frac{\exp \mathbf{g}(\mathbf{x};\boldsymbol{\theta})}{\mathbf{1}^\top \exp \mathbf{g}(\mathbf{x};\boldsymbol{\theta})}, \tag{5}$$

where $\mathbf{1} \in \mathbb{R}^M$ is a vector with all ones. Recall, that ideally $\bar{\mathbf{g}}(\mathbf{x};\boldsymbol{\theta}) \approx \mathbf{e}_\beta$.

The main questions that will be addressed in this paper are:

- How can the uncertainty $\mathrm{Cov}\big(\hat{\boldsymbol{\theta}}_N\big)$ in the parameter estimate be computed efficiently?
- How can the uncertainty in the parameter estimate be propagated to an uncertainty in the classifier output $\hat{\beta}$, i.e., how to estimate $p_{ij} = \mathrm{Prob}\big(\beta = i | \hat{\beta} = j\big)$?
- Can the internal uncertainty be used to detect outliers, e.g., input vectors $\mathbf{x}$ that should not be used in a sensor fusion system when the classification is unreliable?

B. Neural network model structure

A fully connected NN with L layers can be written as

$$\mathbf{h}^{(0)} = \mathbf{x}, \tag{6a}$$

$$\mathbf{a}^{(l+1)} = \begin{pmatrix} \mathbf{h}^{(l)} & 1 \end{pmatrix}^\top W^{(l)}, \quad l = 0, \cdots, L, \tag{6b}$$

$$\mathbf{h}^{(l)} = \sigma\big(\mathbf{a}^{(l)}\big), \quad l = 1, \cdots, L-1, \tag{6c}$$

where $\sigma()$ is an activation function. Collecting all the weights and biases included in the matrices $W^{(l)}$ into the parameter vector

$$\boldsymbol{\theta} \triangleq \begin{bmatrix} \mathrm{Vec}(W^{(L)})^\top & \dots & \mathrm{Vec}(W^{(0)})^\top \end{bmatrix}^\top, \tag{6d}$$

the NN can then be written as a parametric model as

$$\mathbf{g}(\mathbf{x};\boldsymbol{\theta}) = \mathbf{a}^{(L)}. \tag{6e}$$

The dimension of the l:th matrix $W^{(l)}$, is given as $n_{W,l} \times n_{W,l-1} + 1$, where $n_{W,0} = n_x$, $n_{W,L} = M$, where the other $n_{W,l}$ are design choices. The total number of parameters in the model is

$$n_{\boldsymbol{\theta}} = \sum_{l=1}^{L} n_{W,l}(n_{W,l-1} + 1). \tag{7}$$

Usually, the output from the NN is normalized using the softmax function, i.e., the output is given by $\bar{\mathbf{g}}(\mathbf{x};\boldsymbol{\theta})$ in (5). The normalization in (5) forces the output of the NN to be between 0 and 1, and that $\mathbf{1}^\top \bar{\mathbf{g}}(\mathbf{x};\boldsymbol{\theta}) = 1$. Hence $\bar{\mathbf{g}}(\mathbf{x};\boldsymbol{\theta})$ can be interpreted as the relative weight for that the input belongs to the different classes. However, bear in mind that no stochastic assumptions have been made, so the interpretation of this vector to be probabilities should be avoided.

C. Training of the model

Given training data consisting of inputs $\{\mathbf{x}_n\}_{n=1}^N$ with corresponding annotations $\{\mathbf{y}_n\}_{n=1}^N$, a parametric model $\mathbf{g}(\mathbf{x}_n;\boldsymbol{\theta})$ can be trained to estimate a mapping between the input and the true class. Here input $\mathbf{x}_n$ is assumed to be noise free and uncertainty enters only via the annotation of the input $\mathbf{y}_n$, i.e., the measurements. In the literature, see e.g., [26], this is usually done by minimizing the cross-entropy loss function

$$V_N(\boldsymbol{\theta}) = -\sum_{n=1}^{N}\sum_{m=1}^{M} y_{mn} \ln \bar{g}_m(\mathbf{x}_n;\boldsymbol{\theta}), \tag{8a}$$

$$\hat{\boldsymbol{\theta}}_N = \arg\min_{\boldsymbol{\theta}} V_N(\boldsymbol{\theta}). \tag{8b}$$

That is, the cross-entropy is chosen as the metric $d_y()$ in (4). This is motivated by that $\mathbf{y}_n$ can be interpreted as a distribution which the NN should learn, and (8a) is a measure of how close two distributions are to each other [27].

The classification of the input $\mathbf{x}_n$ is commonly done by choosing the class (index) β_n corresponding to the highest value of $\bar{g}_m(\mathbf{x}_n;\hat{\boldsymbol{\theta}}_N)$, i.e.,

$$\hat{\beta}_n = \arg\max_m \bar{g}_m(\mathbf{x}_n;\hat{\boldsymbol{\theta}}_N). \tag{9}$$

III. Quantify uncertainty in the classification

In the literature, many approaches to quantify uncertainty in classification rely on creating ensembles from which a variance is obtained [14, 22, 28–31]. Here, the ensembles are often created by sampling new values of the parameters, which requires sampling from a high dimensional distribution. The proposed method in this paper follows another approach, where the ensembles are created by directly sampling from the output of the NN.

Assume that some parameters $\hat{\boldsymbol{\theta}}_N$ have been found which minimize (8), and that those are the same parameters that would minimize the mean squared error between $\mathbf{g}(\mathbf{x}_n;\hat{\boldsymbol{\theta}}_N)$ and the value before the one-hot encoding denoted $\mathbf{z}_n \in \mathbb{R}^M$. This assumes that it would be possible to measure $\mathbf{z}_n$ (which it is not). Then the relationship between the parametric model $\mathbf{g}(\mathbf{x}_n;\hat{\boldsymbol{\theta}}_N)$ and the measurement can be described as

$$\mathbf{z}_n = \mathbf{g}(\mathbf{x}_n;\hat{\boldsymbol{\theta}}_N) + \mathbf{v}_n, \tag{10}$$

where $\mathbf{v}_n \in \mathbb{R}^M$ is measurement noise which might lead to incorrect annotation. The one-hot encoding can be recovered as

$$y_{mn} = \begin{cases} 1, & \text{if } m = \arg\max_r z_{rn} \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

Recall the assumption of additive noise on $\mathbf{z}$ from (10). Then, given enough training data, the parameters of the model can be assumed Gaussian distributed with mean $\hat{\boldsymbol{\theta}}_N$ and covariance $\mathrm{Cov}(\hat{\boldsymbol{\theta}}_N)$ [16]. Given that the covariance of the parameters $\mathrm{Cov}(\hat{\boldsymbol{\theta}}_N)$ has been computed during the training phase. Using the linearization approach [15], the uncertainty can be propagated to the output as

$$\mathrm{Cov}(\hat{\mathbf{z}}) = \frac{\partial}{\partial \boldsymbol{\theta}} \mathbf{g}(\mathbf{x};\hat{\boldsymbol{\theta}}_N) \mathrm{Cov}(\hat{\boldsymbol{\theta}}_N) \big(\frac{\partial}{\partial \boldsymbol{\theta}} \mathbf{g}(\mathbf{x};\hat{\boldsymbol{\theta}}_N)\big)^\top, \tag{12}$$

for new inputs $\mathbf{x}$. From the Gaussian assumption on the parameters, the predictions before the softmax layer can be assumed to be M dimensional Gaussian distributed,

$$\mathbf{z} \sim \mathcal{N}(\hat{\mathbf{z}}, \mathrm{Cov}(\hat{\mathbf{z}})). \tag{13}$$

Here, an estimate of $\mathbf{z}$ for the input $\mathbf{x}$ is given by $\hat{\mathbf{z}} = \mathbf{g}(\mathbf{x}, \hat{\boldsymbol{\theta}}_N)$. In an MC-like fashion, per input $\mathbf{x}$, an ensemble of predictions before the softmax layer can be created by sampling K values from (13), and collecting them in the ensemble

$$Z = \begin{bmatrix} \mathbf{z}^1 & \dots & \mathbf{z}^K \end{bmatrix} \in \mathbb{R}^{M \times K}. \tag{14}$$

To obtain a classification of $\mathbf{x}$ which takes the uncertainty into consideration, first find the class corresponding to the max value per sample in the ensemble and collect them into a vector. This vector is summarizing how many of the samples predicted a specific class. Secondly, normalize this vector with the number of samples in the ensemble. That is, for each input $\mathbf{x}$, and for every class m compute

$$\tilde{p}_m = \frac{1}{K} \sum_{k=1}^{K} \mathbb{1}(\tilde{\beta}_k = m), \tag{15a}$$

$$\tilde{\beta}_k = \arg\max_i Z_{ik}. \tag{15b}$$

Here Z_{mk} is the m:th value of the k:th sample, $\tilde{\beta}_k$ is the classified label for the k:th sample, and $\mathbb{1}()$ is the indicator function. Notice that $\tilde{\mathbf{p}} \in \mathbb{I}^M$ can be interpreted as an estimate of the probability mass function (PMF) for a given input.

Even NNs which are considered to be small, i.e., with few weights and biases, often have hundreds of thousands (if not millions) of parameters. This might result in high computation demands when inverting an $n_{\boldsymbol{\theta}} \times n_{\boldsymbol{\theta}}$ matrix to compute the covariance matrix. Further, the amount of data to be stored might be larger than the available memory capacity. To make the proposed method computationally tractable, only the parameters in the q last layers $\boldsymbol{\theta}^q \in \mathbb{R}^{n_q}$ are considered to be parameters in the parametric model, where n_q is the number of parameters given by (7) but where the sum starts from $l = L - q + 1$ instead of zero. The parameters in the earlier layers $\boldsymbol{\xi} \in \mathbb{R}^{n_{\boldsymbol{\theta}} - n_q}$ are assumed to be fixed and used to construct high-level features (basis function) of the input, from here on denoted $\boldsymbol{\psi}(\mathbf{x}; \boldsymbol{\xi}) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_{\boldsymbol{\theta}} - n_q} \to \mathbb{R}^{n_{W,L-q+1}}$. For example

$$\mathbf{g}(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{g}^q(\boldsymbol{\psi}(\mathbf{x}; \boldsymbol{\xi}); \boldsymbol{\theta}^q), \tag{16a}$$

i.e., $\mathbf{g}^q(\boldsymbol{\psi}(\mathbf{x}; \boldsymbol{\xi}); \boldsymbol{\theta}^q) : \mathbb{R}^{n_{W,L-q+1}} \times \mathbb{R}^{n_q} \to \mathbb{R}^M$ where

$$\boldsymbol{\psi}(\mathbf{x}; \boldsymbol{\xi}) = [\mathbf{h}^{(L-q+1)}, \quad 1]. \tag{16b}$$

From here on, with a bit abuse of notation, $\mathbf{x}$ denotes the high-level features $\boldsymbol{\psi}(\mathbf{x}; \boldsymbol{\xi})$, $\mathbf{g}(\mathbf{x}; \boldsymbol{\theta})$ denotes the parametric model given those features $\mathbf{g}^q(\boldsymbol{\psi}(\mathbf{x}; \boldsymbol{\xi}); \boldsymbol{\theta}^q)$, and $\boldsymbol{\theta}$ denotes the parameters $\boldsymbol{\theta}^q$. The choice of high-level features is arbitrary, but independent of how this choice is made, the forthcoming analysis and the proposed method to quantify uncertainty in the classification are still valid.

This simplification has many similarities with transfer-learning and extreme learning machines, and can be motivated by their success. In transfer-learning, the high-level features created from training on one task are reused for a second one, where only the parameters in the last layers are tuned for the second task [26, 32]. While in extreme learning machines it is shown that as long as you have enough random high-level features one can combine them to obtain almost perfect accuracy in the prediction [33].

IV. Recursive update of the parameter covariance

The proposed method to quantify uncertainty in the classification require to compute the parameter covariance of the NN during the training phase. Given N samples, it can be shown that one approximation of the covariance of the parameters is given by

$$\operatorname{Cov}(\hat{\boldsymbol{\theta}}_N) = P_N^{\boldsymbol{\theta}}, \tag{17a}$$

$$\left(P_N^{\boldsymbol{\theta}}\right)^{-1} = \sum_{n=1}^{N} \sum_{m=1}^{M} \mathbf{u}_{mn}^\top \mathbf{u}_{mn}, \tag{17b}$$

$$\mathbf{u}_{mn} \triangleq \sqrt{\bar{g}_m(\mathbf{x}_n; \hat{\boldsymbol{\theta}}_N)(1 - \bar{g}_m(\mathbf{x}_n; \hat{\boldsymbol{\theta}}_N))} \left(\frac{\partial g_m(\mathbf{x}_n; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right), \tag{17c}$$

where (17b) is the second derivative of the loss function in (8a) with respect to the parameters $\boldsymbol{\theta}$, and $\mathbf{u}_{mn} \in \mathbb{R}^{1 \times n_{\boldsymbol{\theta}}}$ is defined such that the second derivative of the loss function can be written in a quadratic form. Note that to compute $\mathbf{u}_{mn}$, only the first derivative of the loss function is required. The parameter covariance can be computed cheaply during the training phase of the NN, where $\mathbf{u}_{mn}$ is obtained by some efficient implementation of automatic differentiation, e.g., backpropagation. A useful property of this approximation of the covariance is that it can recursively be updated given new measurements. This can be formalized as follows.

Theorem IV.1. *Given the structure of the covariance matrix as in* (17)*. Assume that the covariance matrix has been computed for the n:th first measurements, $P_n^{\boldsymbol{\theta}}$, and that the matrix has full rank. Then given a new measurement of $\mathbf{u}_{mn}$, the covariance matrix can recursively be updated as*

$$P_{n+1}^{\boldsymbol{\theta}} = P_n^{\boldsymbol{\theta}} - P_n^{\boldsymbol{\theta}} (U_{n+1})^\top \left(I_M + U_{n+1} P_n^{\boldsymbol{\theta}} (U_{n+1})^\top\right)^{-1} U_{n+1} P_n^{\boldsymbol{\theta}}, \tag{18}$$

where I_r denotes the identity matrix of size r and

$$U_n = \begin{bmatrix} \mathbf{u}_{1n}^\top & \dots & \mathbf{u}_{Mn}^\top \end{bmatrix}^\top \in \mathbb{R}^{M \times n_{\boldsymbol{\theta}}}. \tag{19}$$

Proof. Rewrite (17a) for $n+1$ measurements as

$$P_{n+1}^{\boldsymbol{\theta}} = \left(\sum_{i=1}^{n+1} (U_i)^\top U_i \right)^{-1} \tag{20a}$$

$$= \left(\sum_{i=1}^{n} (U_i)^\top U_i + (U_{n+1})^\top U_{n+1} \right)^{-1} \tag{20b}$$

$$= \left(\left(P_n^{\boldsymbol{\theta}}\right)^{-1} + (U_{n+1})^\top U_{n+1} \right)^{-1}. \tag{20c}$$

where U_n is given by (19). With the assumption of $P_n^{\boldsymbol{\theta}}$ has full rank, it is easy to show, using Woodbury matrix identity, that (20) can be written as (18). □

Note that if $P_n^{\boldsymbol{\theta}} = 0$ then $P_{n+1}^{\boldsymbol{\theta}} = 0 \ \forall n$. To make sure this does not happens to quickly it is required to initialize $P_0^{\boldsymbol{\theta}}$ as a matrix with full rank, this also ensure positive definiteness

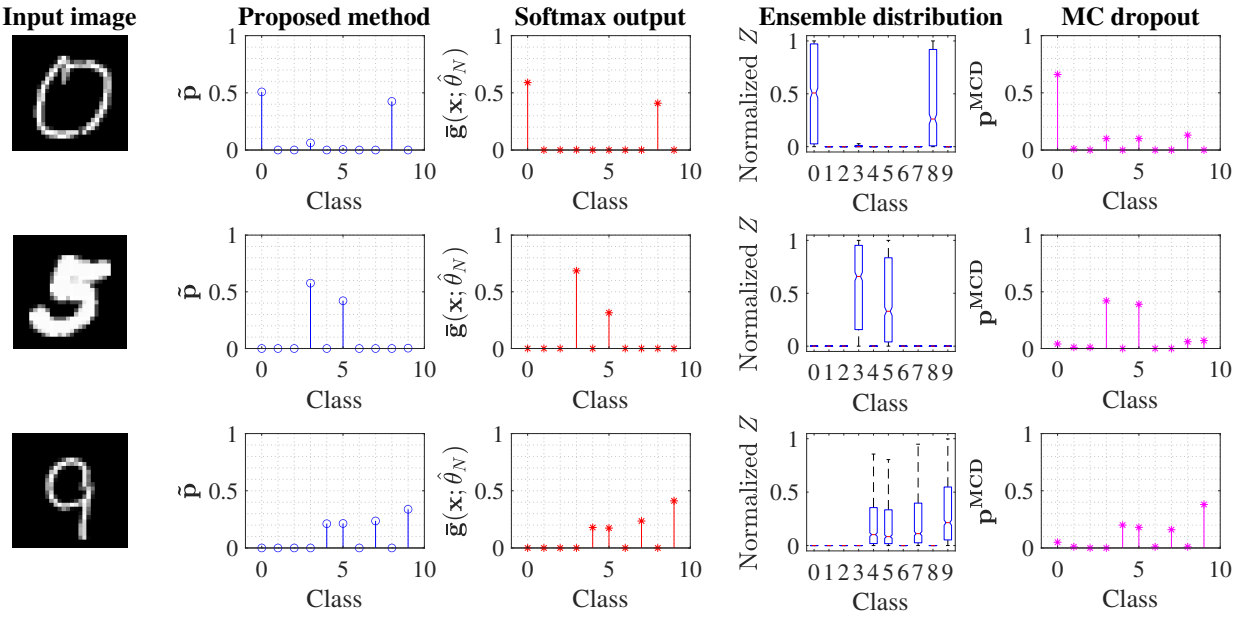


Fig. 2: Images of handwritten digits with their classification where uncertainty is included. Classification including uncertainty using the proposed method $\tilde{\mathbf{p}}$ (see Section III), the output of the softmax $\bar{\mathbf{g}}(\mathbf{x};\boldsymbol{\theta})$, and distribution of the ensemble Z after normalization using the softmax function illustrated with a boxplot, is shown. To compare to the proposed method, classification including uncertainty using MC dropout (see Section V-B) $\mathbf{p}^{\mathbf{MCD}}$ is also shown.

of the recursive updates. For example $P_0^{\boldsymbol{\theta}} = \alpha^{-1} I_{n_{\boldsymbol{\theta}}}$ can be used, where α is a design choice.

V. Experimental study

To evaluate the proposed method, an NN was trained to classify handwritten digits where the images came from the classical MNIST dataset [34]. The NN had three hidden layers where $n_{W,0} = 784$, $n_{W,1} = 784$, $n_{W,2} = 200$, $n_{W,3} = 100$, and $n_{W,4} = 10$, hence $n_{\boldsymbol{\theta}} = 793550$. Dropout was used during training, where the weights of biases in the second and third layer were dropped with probability 0.1. This so that the proposed method can be compared with MC dropout. Rectified linear unit (ReLU) was used as activation function and output was normalized using the softmax function. To minimize (8), the ADAM optimizer [35] was used with the standard settings. The NN was trained for 3 epochs, had an initial learning rate of 10^{-4}, and l^2 regularization of 10^{-4}. The accuracy of the NN on validation data was 90.66%. To decrease the dimension of the parameter covariance, all parameters from the layers before the last layer were considered to be high-level features, i.e., $q = 1$ in (16). Hence the parametric model had $n_{\boldsymbol{\theta}} = 1010$ parameters. The images shown in Fig. 2 are not how the classification of images commonly looks like, these images are chosen since their classification done by the NN is uncertain.

A. Classification with uncertainty: Proposed method

Using the proposed method from Section III, for a couple of images of handwritten digits, classification where uncertainty was taken into consideration $\tilde{\mathbf{p}}$ are shown in Fig. 2. Compared with the output of the softmax layer $\bar{\mathbf{g}}(\mathbf{x};\hat{\boldsymbol{\theta}}_N)$, the emphasis is changed between certain and uncertain classes, i.e., the probability mass is smeared out for $\tilde{\mathbf{p}}$ compared to $\bar{\mathbf{g}}(\mathbf{x};\hat{\boldsymbol{\theta}}_N)$. This can indicate that classification using $\bar{\mathbf{g}}(\mathbf{x};\hat{\boldsymbol{\theta}}_N)$ underestimates the uncertainty in the decision. This is especially clear for the misclassified image where the inclusion of the uncertainty almost results in the correct classification of the image. In Fig. 2 the distribution for the ensemble Z after normalization using the softmax function is also visualized using a boxplot.

B. Classification with uncertainty: Monte Carlo Dropout

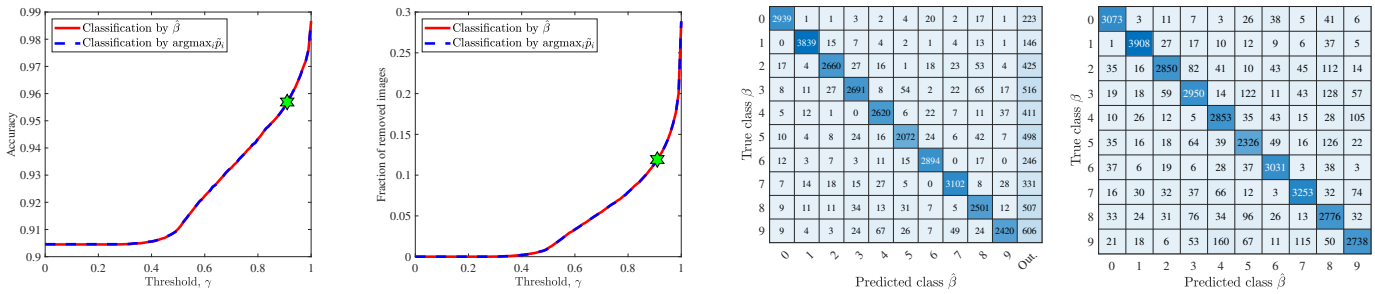
In the literature, MC dropout presented in [14], is a method that is often used in quantifying the uncertainty in the output of NNs. Dropout is a regularization method where parameters are "*dropped*" with a certain probability during training. Hence to train a NN with dropout can be interpreted as training an ensemble of thinner networks [36]. The idea of MC dropout is to use dropout to create an ensemble of classifications Z^{MCD} during the validation phase. From this ensemble, similarly to (15), one could create a classification of the input where uncertainty is taken into consideration. That is

$$p_m^{MCD} = \frac{1}{K}\sum_{k=1}^{K} \mathbb{1}(\beta_k^{MCD} = m), \tag{21a}$$

$$\beta_k^{MCD} = \arg\max_i Z_{ik}^{MCD}, \tag{21b}$$

hence $\mathbf{p}^{\mathbf{MCD}} \in \mathbb{I}^M$. Here Z_{mk}^{MCD} is the m:th value of the k:th sample in the ensemble created by MC dropout, and β_k^{MCD} is the classified label for the k:th sample.

Fig. 2 shows the classification of a couple of images where MC dropout is used to quantify the uncertainty in the classification. By visual inspection, the proposed method and MC dropout seem to be on par with each other. Compared to the proposed method, MC dropout requires sampling



(a) Accuracy. (b) Fraction removed. (c) With outlier class. (d) All images.

Fig. 3: Detection of images with uncertain classification. These are put into an artificial outlier class, i.e., the condition (22) with $\tilde{\mathbf{p}}$ given by (15a). In (a) the accuracy of the images not in the outlier class, and in (b) the fraction of images ending up in that class (the fraction of images removed) are plotted as a function of the threshold γ. In (c) the confusion matrix for images including an outlier class is shown for $\gamma = 0.9$ (the green hexagon in Fig. 3a and Fig. 3b). This compared to (d) where the confusion matrix for all images is shown.

from a distribution with the same dimension as the number of parameters, while the proposed method samples from a distribution with the same size as the number of classes, which usually is significantly lower. This can be seen in the computation time. Using the author's implementation, creating an ensemble consisting of 100 samples using MC dropout took a bit more time than one hour while creating an ensemble of 1000 samples using the proposed method took around 60 seconds.

C. Detection of uncertain classifications

Having access to uncertainty in the classification, it is possible to detect which images the NN have trouble to classify. If the most of the probability mass of $\tilde{\mathbf{p}}$ is concentrated at one class, the NN is assumed to be certain in its classification. Using this methodology, an image with an uncertain label can be found by comparing the max-value of $\tilde{\mathbf{p}}$ to a threshold γ. That is, the classification is uncertain if

$$\max \tilde{\mathbf{p}} < \gamma. \tag{22}$$

This makes it possible to collect all images with uncertain classification into a new artificial outlier class, which can be handled separately.

In Fig. 3 the detection of uncertain images is shown. The uncertain images are put into an artificial outlier class containing images that cannot be classified with high confidence. The accuracy for images not in the outlier class, and how large a fraction of the images ends up in the outlier class are plotted against the threshold in Fig. 3a and Fig. 3b, respectively. This is done using the output max index of the softmax layer $\hat{\beta}$ (red) and the $\arg\max$ of $\tilde{\mathbf{p}}$ (blue). For example, one can see that with only removing a small percentage of the images, the accuracy is boosted to almost perfect classification.

The confusion matrix is often used to measure how well a classifier works. It shows how different classes get mixed up, i.e., the probability of detection is given by dividing the diagonal element with the column sum of the matrix. In Fig. 3c, the confusion matrix with an additional outlier class included is shown when $\gamma = 0.9$. Compared to the confusion matrix for all images seen in Fig. 3d, it is clear that the majority of the images in the outlier class are wrongly classified.

VI. Conclusion

This paper proposes a method to quantify uncertainty in classification tasks where NNs are used as the model. The method is based upon linearization to propagate uncertainty in the parameters of the NN to uncertainty of the classification. The uncertainty in the parameters is computed during the training phase. An approach on how to recursively compute the covariance of the parameters, which is required for the proposed method, is also suggested (see Theorem IV.1). The paper also presents a method how to find images with an uncertain classification that could be handled separately in a decision-making process, i.e., detect outliers.

There are several interesting directions left to explore regarding the quantification of uncertainty in the decisions made by NNs. For example, what is the lowest covariance one could expect the parameters to have, how could prior information be included in the decision, and how to utilize this in a sensor fusion framework to guarantee safety? These are examples of relevant directions for future work.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Adv. in Neural Inf. Process. Syst. (NIPS) 25*, Lake Tahoe, NVn USA, 2012, pp. 1097–1105, 3-8, Dec.

[2] I. Goodfellow *et al.*, "Generative adversarial nets," in *Adv. in Neural Inf. Process. Syst. (NIPS) 28*, Montréal, QC, Canada, 2014, pp. 2672–2680, 8-13 Dec.

[3] Q. Li *et al.*, "Deep neural networks for improved, impromptu trajectory tracking of quadrotors," in *Proc. of IEEE Int. Conf. on Robot. and Autom. (ICRA)*. Singapore, Singapore: IEEE, 2017, pp. 5183–5189, 19 May–3 June.

[4] B. Karg and S. Lucia, "Efficient representation and approximation of model predictive control laws via deep learning," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3866–3878, Jan. 2020.

[5] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *J. of Field Robotics*, vol. 37, no. 3, pp. 362–386, Jan. 2020.

[6] S. A. Bagloee, M. Tavana, M. Asadi, and T. Oliver, "Autonomous vehicles: challenges, opportunities, and future implications for transportation policies," *J. of Modern Trans.*, vol. 24, no. 4, pp. 284–303, Dec. 2016.

[7] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: a survey of case studies," *Adv. in Neural Inf. Process.*

Syst. (NIPS) 34 Workshop: ML Retrospectives, Surveys & Meta-Analyses (ML-RSA), vol. 33, 7–12 Dec 2020.

[8] A. D'Amour *et al.*, "Underspecification presents challenges for credibility in modern machine learning," *arXiv preprint arXiv:2011.03395*, 2020.

[9] F. K. Gustafsson, M. Danelljan, G. Bhat, and T. B. Schön, "Energy-Based Models for Deep Probabilistic Regression," in *Proc. of 16th European Conf. on Comput. Vision (ECCV)*, Glasgow, UK/Online, 2020, pp. 325–343, 23-28 Aug.

[10] A. Kendall and Y. Gal, "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" in *Adv. in Neural Inf. Process. Syst. (NIPS) 31*. Curran Associates, Inc., 4–9 Dec 2017, pp. 5574–5584, Long Beach, CA, USA, 4–9 Dec.

[11] Z. Ghahramani, "Probabilistic machine learning and artificial intelligence." *Nature*, vol. 521, no. 7553, pp. 452 – 459, Feb. 2015.

[12] K. Patel and S. Waslander, "Accurate prediction and uncertainty estimation using decoupled prediction interval networks," *arXiv preprint arXiv:2202.09664*, 2022.

[13] NTSB, "Preliminary Report Highway HWY18MH010," National Transportation Safty Board (NTSB), Technical Specification (TS), 05 2018. [Online]. Available: https://data.ntsb.gov/Docket/?NTSBNumber=HWY18MH010

[14] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," in *Proc. of the 33td Int. Conf. on Mach. Learn. (ICML).*, New York, NY, USA, 2016, pp. 1050–1059, 20–22 Jun.

[15] M. Malmström, "Uncertainties in neural networks a system identification approach," Licentiate Thesis, Dept. Elect. Eng., Linköping University, Linköping, Sweden, Apr 2021.

[16] L. Ljung, *System identification: theory for the user (2nd edition)*. PTR Prentice Hall: Upper Saddle River, NJ, USA, 1999.

[17] M. Malmström, I. Skog, D. Axehill, and F. Gustafsson, "Modeling of the tire-road friction using neural networks including quantification of the prediction uncertainty," in *Proc. of IEEE 24th Int. Conf. on Inf. Fusion (FUSION)*. Sun City, South Africa/ Virtual: IEEE, 2021, pp. 1–6, Nov 1-4.

[18] J. T. G. Hwang and A. A. Ding, "Prediction Intervals for Artificial Neural Networks," *J. Am. Stat. Assoc. (JSTOR)*, vol. 92, no. 438, pp. 748–757, 1997.

[19] I. Rivals and L. Personnaz, "Construction of confidence intervals for neural networks based on least squares estimation," *Elsevier J. Neural Netw.*, vol. 13, pp. 463–484, Jan. 2000.

[20] G. Papadopoulos, P. Edwards, and A. Murray, "Confidence estimation methods for neural networks: a practical comparison," *IEEE Trans. Neural Netw.*, vol. 12, pp. 1278–1287, Nov. 2001.

[21] G. Chryssoloiuris, M. Lee, and A. Ramsey, "Confidence interval prediction for neural network models," *IEEE Trans. Neural Netw.*, vol. 7, pp. 229–232, Jan. 1996.

[22] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Adv. in Neural Inf. Process. Syst. (NIPS) 31*. Curran Associates, Inc., 4–9 Dec 2017, Long Beach, CA, USA, 4–9 Dec.

[23] P. W. Koh and P. Liang, "Understanding Black-box Predictions via Influence Functions," in *Proc. of the 34th Int. Conf. on Mach. Learn. (ICML).*, Sydney, Australia, 2017, pp. 1885–1894, 06–11 Aug.

[24] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks." in *Proc. of 5th Int. Conf. for Learn. Representations (ICLR)*, Toulon, France, 2016, 24-26 Apr.

[25] R. Girshick, "Fast R-CNN," in *Proc. of IEEE Int. Conf. on Comput. Vision (ICCV)*, Araucano Park, Las Condes, Chile, 2015, pp. 1440–1448, 11-18, Dec.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016, London, England.

[27] M. A. Nielsen, *Neural networks and deep learning*. Determination press San Francisco, CA, 2015, vol. 25.

[28] G. Carannante, N. C. Bouaynaya, and L. Mihaylova, "An enhanced particle filter for uncertainty quantification in neural networks," in *Proc. of IEEE 24th Int. Conf. on Inf. Fusion (FUSION)*. Sun City, South Africa/ Virtual: IEEE, 2021, pp. 1–7, Nov 1-4.

[29] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight Uncertainty in Neural Networks," in *Proc. of the 32nd Int. Conf. on Mach. Learn. (ICML).*, Lille, France, 2015, pp. 1613–1622, 6–11 Jul.

[30] E. Ilg *et al.*, "Uncertainty estimates and multi-hypotheses networks for optical flow," in *Proc. of 15th European Conf. on Comput. Vision (ECCV)*, Munich, Germany, 2018, pp. 652–667, 8-14 Sep.

[31] M. Teye, H. Azizpour, and K. Smith, "Bayesian Uncertainty Estimation for Batch Normalized Deep Networks," in *Proc. of the 35th Int. Conf. on Mach. Learn. (ICML).*, 6–11 Jul 2018, pp. 4907–4916, Stockholm, Sweden ,6–11 Jul.

[32] Y. Bengio, "Deep learning of representations for unsupervised and transfer learning," in *Proc. of the 29th Int. Conf. on Mach. Learn. (ICML). Workshop on Unsupervised and Transfer Learn.* Edinburgh, Scotland, UK: JMLR Workshop and Conference Proceedings, 2012, pp. 17–36, 26 June–1 Jul.

[33] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, Dec. 2006.

[34] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[35] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization." in *Proc. of 3rd Int. Conf. for Learn. Representations (ICLR)*, 7–9 May 2015, San Diego, CA, USA.

[36] N. Srivastava *et al.*, "Dropout: A simple way to prevent neural networks from overfitting," *The J. of Mach. Learn. Research (JMLR)*, no. 2, p. 1929, 2015.