Encoding the latent posterior of Bayesian Neural Networks for uncertainty quantification

Gianni Franchi, Andrei Bursuc, Emanuel Aldea, Séverine Dubuisson, Isabelle Bloch

Encoding the latent posterior of Bayesian Neural Networks for uncertainty quantification

Gianni Franchi
ENSTA Paris
Institut polytechnique de Paris
gianni.franchi@ensta-paris.fr

Andrei Bursuc
valeo.ai
andrei.bursuc@valeo.com

Emanuel Aldea
SATIE, Université Paris-Sud
Université Paris-Saclay
emanuel.aldea@u-psud.fr

Séverine Dubuisson
CNRS, LIS
Aix Marseille University
severine.dubuisson@lis-lab.fr

Isabelle Bloch
LTCI, Télécom Paris
Institut polytechnique de Paris
isabelle.bloch@telecom-paris.fr

Abstract

Bayesian neural networks (BNNs) have been long considered an ideal, yet unscalable solution for improving the robustness and the predictive uncertainty of deep neural networks. While they could capture more accurately the posterior distribution of the network parameters, most BNN approaches are either limited to small networks or rely on constraining assumptions such as parameter independence. These drawbacks have enabled prominence of simple, but computationally heavy approaches such as Deep Ensembles, whose training and testing costs increase linearly with the number of networks. In this work we aim for efficient deep BNNs amenable to complex computer vision architectures, e.g. ResNet50 DeepLabV3+, and tasks, e.g. semantic segmentation, with fewer assumptions on the parameters. We achieve this by leveraging variational autoencoders (VAEs) to learn the interaction and the latent distribution of the parameters at each network layer. Our approach, Latent-Posterior BNN (LP-BNN), is compatible with the recent BatchEnsemble method, leading to highly efficient (in terms of computation and memory during both training and testing) ensembles. LP-BNNs attain competitive results across multiple metrics in several challenging benchmarks for image classification, semantic segmentation and out-of-distribution detection.

1. Introduction

Most top-performing approaches for predictive uncertainty estimation with Deep Neural Networks (DNNs) [1, 33, 11] are essentially based on ensembles [27] which have been shown to display many strengths: stability, mode diversity, *etc*. [9]. However their notable drawback in terms of computational cost often make them prohibitive. The study of DNN uncertainty is particularly challenging also due to the potentially fuzzy definition of in- and out-of-distribution samples, or to the difficulty in identifying the type of uncertainty behind wrong predictions. However, in the context of potential widespread adoption of DNNs for practical applications, uncertainty estimation is essential for computer vision tasks, on par with the traditional goal of reaching high accuracy.

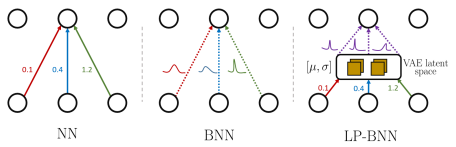

Fig. 1: In a standard NN each weight has a fixed value. In a BNN [3] all weights follow Gaussian distributions and are assumed to be mutually independent: each weight is factorized by a Gaussian distribution. LP-BNN considers that the weights at every layer follow a multivariate Gaussian distribution, with a latent weight space composed of independent Gaussian distributions. This enables computing weight distributions in a lower dimensional space.

In this work we address uncertainty estimation with BNNs, which propose an intuitive and elegant formalism suited for this task. For DNNs, we often assume that the network weights to be modeled as random variables following a Gaussian distribution, and that BNNs seek to estimate the posterior distribution of the weights. In most BNN variants however the weights are assumed to be independent of each other. While this assumption ensures computational tractability, it can be damaging as a more complex organization can emerge within network layers, and that higher

level correlations contribute to better performance and generalization [47, 45, 2]. Yet, even under such settings, BNNs are often challenging to train at scale [7]. In response, many approaches [33, 11] have advanced efficient approaches to estimate the posterior distribution, yet also building on similar assumptions about the training process and the independence of the weights of the DNN. In this work, we propose to estimate the posterior of a BNN with inter-weight correlations, in a stable and computationally efficient manner. So far BNNs have failed to reach competitive performance in popular computer vision benchmarks, often lagging behind Deep Ensemble [27]. To the best of our knowledge, this is the first BNN approach that can cope with complex models and achieve state-of-the-art results for computer vision tasks without the simplifying weight independence assumption.

In this paper, we advance a different deep BNN model, named *Latent Posterior BNN* (LP-BNN), for which the posterior distribution of the weights at every layer is encoded with a variational autoencoder (VAE) [25] into a latent space that follows a Gaussian distribution. This makes it possible to efficiently learn weight correlations via the VAE. Our training procedure may be viewed as a standard optimization problem involving the training of multiple VAEs, as opposed to specific BNN training strategies requiring individual weight samplings whose variance hinders training stability [7]. While VAEs are typically used to learn distribution in order to generate images, here our aim is to learn the distribution that generates DNNs to reach to a higher accuracy. We outline LP-BNN in Figure 1.

2. Background

In this section, we present the chosen formalism for this work and offer a short background on BNNs.

2.1. Preliminaries

We consider a training dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with n samples and labels, corresponding to two random variables $X \sim \mathcal{P}_X$ and $Y \sim \mathcal{P}_Y$. Without loss of generality we represent $\mathbf{x}_i \in \mathbb{R}^d$ as a vector, and y_i as a scalar label. We process the input data $\mathbf{x}_i$ with a neural network $f_\Theta(\cdot)$ with parameters Θ, that outputs a classification or regression prediction. We view the neural network as a probabilistic model with $f_\Theta(\mathbf{x}_i) = P(Y = y_i \mid X = \mathbf{x}_i, \Theta)$. In the following, when there are no ambiguities, we discard the random variable from notations. For classification, $P(y_i \mid \mathbf{x}_i, \Theta)$ is a categorical distribution over the set of classes over the domain of Y, typically corresponding to the cross-entropy loss function, while for regression $P(y_i \mid \mathbf{x}_i, \Theta)$ is a Gaussian distribution of real values over the domain of Y when using the squared loss function. For simplicity we unroll our reasoning for the classification task.

In supervised learning, we leverage gradient descent for learning Θ that minimizes the cross-entropy loss, which is equivalent to finding the parameters that maximize the likelihood estimation (MLE) $P(\mathcal{D} \mid \Theta)$ over the training set $\Theta_{\text{MLE}} = \arg\max_\Theta \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} \log P(y_i \mid \mathbf{x}_i, \Theta)$, or equivalently minimize the following loss function:

$$\mathcal{L}_{\text{MLE}}(\Theta) = -\sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} \log P(y_i \mid \mathbf{x}_i, \Theta). \tag{1}$$

The Bayesian approach enables adding a prior information on the parameters Θ, by placing a prior distribution $\mathcal{P}(\Theta)$ upon them. This prior represents some expert knowledge about the dataset and the model. Instead of maximizing the likelihood, we can now find the maximum a posteriori (MAP) weights for $\mathcal{P}(\Theta \mid \mathcal{D})$ decomposed into $\mathcal{P}(\mathcal{D} \mid \Theta)$ and $\mathcal{P}(\Theta)$ to compute $\Theta_{\text{MAP}} = \arg\max_\Theta \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} \log \mathcal{P}(y_i \mid \mathbf{x}_i, \Theta) + \log \mathcal{P}(\Theta)$, *i.e.* to minimize the following loss function:

$$\mathcal{L}_{\text{MAP}}(\Theta) = -\sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} \log P(y_i \mid \mathbf{x}_i, \Theta) - \log P(\Theta), \tag{2}$$

inducing a specific distribution over the functions computed by the network and a regularization of the weights. For a Gaussian prior, Eq. (2) reads as L_2 regularization (weight decay).

2.2. Bayesian Neural Networks

In most neural networks only the Θ_{MAP} weights computed during training are kept for predictions. Conversely, in BNNs we aim to find the posterior distribution of the parameters given the training dataset $P(\Theta \mid \mathcal{D})$, not only the values corresponding to the MAP. Here we can make a prediction y on a new sample $\mathbf{x}$ by computing the expectation of the predictions from an infinite ensemble corresponding to different configurations of the weights sampled from the posterior:

$$P(y \mid \mathbf{x}, \mathcal{D}) = \int P(y \mid \mathbf{x}, \Theta) P(\Theta \mid \mathcal{D}) d\Theta, \tag{3}$$

which is also known as Bayes ensemble. The integral in Eq. (3), which is calculated over the domain of Θ, is intractable, and in practice it is approximated by averaging predictions from a limited set $\{\Theta_1, \dots \Theta_J\}$ of J weight configurations sampled from the posterior distribution:

$$P(y \mid \mathbf{x}, \mathcal{D}) \approx \frac{1}{J} \sum_{j=1}^{J} P(y \mid \mathbf{x}, \Theta_j). \tag{4}$$

Although BNNs are elegant and easy to formulate, their inference is non-trivial and has been subject to extensive research across the years [20, 31, 37]. Early approaches relied on Markov chain Monte Carlo variants for inference, while progress in variational inference (VI) [24] has enabled

a recent revival of BNNs [3, 14]. VI turns posterior inference into an optimization problem. In detail, VI finds the parameters ν of a distribution $Q_\nu(\Theta)$ on the weights that approximates the true Bayesian posterior distribution of the weights $P(\Theta \mid \mathcal{D})$ through KL-divergence minimization. This is equivalent to minimizing the following loss function, also known as expected lower bound (ELBO) loss [3, 25]:

$$\mathcal{L}_{\text{BNN}}(\Theta, \nu) = - \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} \mathbb{E}_{\Theta \sim Q_\nu(\Theta)} \log \left(P(y_i \mid \mathbf{x}_i, \Theta) \right) + \text{KL}(Q_\nu(\Theta) || P(\Theta)). \quad (5)$$

The loss $\mathcal{L}_{\text{BNN}}(\Theta, \nu)$ is composed of two terms: the KL term depends on the weights and the prior $P(\Theta)$, while the likelihood term is data dependent. This loss strives to simultaneously capture faithfully the complexity and diversity of the information from data $\mathcal{D}$, while preserving the simplicity of the prior $P(\Theta)$. To optimize this loss, Blundell *et al*. [3] proposed leveraging the *re-parameterization trick* [25, 44], foregoing the expensive MC estimates.

Discussion. BNNs are particularly appealing for uncertainty quantification thanks to the ensemble of predictions from multiple weight configurations sampled from the posterior distribution. However this brings an increased computational and memory cost. For instance, the simplest variant of BNNs with fully factorized Gaussian approximation distributions [3, 14], *i.e*. each weight consists of a Gaussian mean and variance, carries a double amount of parameters. In addition, Dusenberry *et al*. [7] pointed our that BNNs often underfit, and need multiple tunings to stabilize training dynamics involved by the loss function and the variance from weight samplings at each forward pass. Due to computational limitations, most BNN approaches assume that parameters are not correlated. This hinders their effectiveness, as empirical evidence has shown that encouraging weight collaboration improves training stability and generalization [43, 45, 47].

In order to calculate a tractable weight correlation aware posterior distribution, we propose to leverage a VAE to compute compressed latent distributions from which we can sample new weight configurations. We rely on the recent BatchEnsemble (BE) method [50] to further improve the parameter-efficiency of BNNs. We now proceed to describe BE and then derive our approach.

2.3. BatchEnsemble

Deep Ensembles (DEs) [27] are a popular and pragmatic alternative to BNNs. While DEs boast outstanding accuracy and predictive uncertainty, their training and testing cost increases linearly with the number of networks. This drawback has motivated the emergence of a recent stream of works proposing efficient ensemble methods [1, 11, 33, 35, 50]. One of the most promising ones is BatchEnsemble [50]

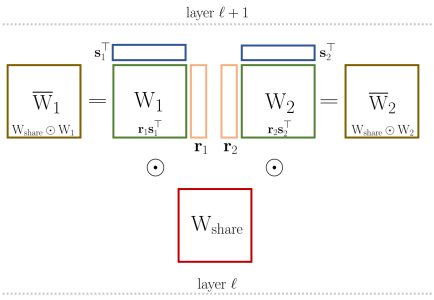

Figure 2: Illustration on how BatchEnsemble generates the ensemble weights for an ensemble of size $J = 2$.

which mimics in a parameter-efficient manner one of the main strengths of DE, *i.e*. diverse predictions [9].

In a nutshell, BE builds up an ensemble from a single base network (shared among ensemble members) and a set of layer-wise weight matrices specific to each member. At each layer, the weight of each ensemble member is generated from the Hadamard product between a weight shared among all ensemble members, called *"slow weight"*, and a Rank-1 matrix that varies among all members, called *"fast weight"*. Formally, let $\mathbf{W}_{\text{share}} \in \mathbb{R}^{m \times p}$ be the slow weights in a neural network layer with input dimension m and with p outputs. Each member j from an ensemble of size J owns a fast weight matrix $\mathbf{W}_j \in \mathbb{R}^{m \times p}$. $\mathbf{W}_j$ is a Rank-1 matrix computed from a tuple of trainable vectors $\mathbf{r}_j \in \mathbb{R}^m$ and $\mathbf{s}_j \in \mathbb{R}^p$, with $\mathbf{W}_j = \mathbf{r}_j \mathbf{s}_j^\top$. BE generates from them a family of ensemble weights as follows: $\overline{\mathbf{W}}_j = \mathbf{W}_{\text{share}} \odot \mathbf{W}_j$, where $\odot$ is the Hadamard product. Each $\overline{\mathbf{W}}_j$ member of the ensemble is essentially a Rank-1 perturbation of the shared weights $\mathbf{W}_{\text{share}}$ (see Figure 2). The sequence of operations during the forward pass reads:

$$h = a\left((W_{\text{share}}^\top (\mathbf{x} \odot \mathbf{s}_j)) \odot \mathbf{r}_j \right), \quad (6)$$

where a is an activation function and h the output activations.

The operations in BE can be efficiently vectorized, enabling each member to process in parallel the corresponding subset of samples from the mini-batch. $\mathbf{W}_{\text{share}}$ is trained in a standard manner over all samples in the mini-batch. A BE network $f_{\Theta^{\text{BE}}}$ is parameterized by an extended set of parameters $\Theta^{\text{BE}} = \left\{ \theta^{\text{slow}} : \{\mathbf{W}_{\text{share}}\}, \theta^{\text{fast}} : \{\mathbf{r}_j, \mathbf{s}_j\}_{j=1}^J \right\}$.

With its multiple sub-networks parameterized by a reduced set of weights, BE is a practical method that can potentially improve the scalability of BNNs. We take advantage of the small size of the fast weights to capture efficiently the interactions between units and to compute a latent distribution of the weights. We detail our approach below.

3. Efficient Bayesian Neural Networks (BNNs)

3.1. Encoding the posterior weight distribution of a BNN

Most BNN variants assume full independence between weights both inter- and intra-layer. Modeling precisely weight correlations in modern high capacity DNNs with thousands to millions of parameters per layer [16] is however a daunting endeavor due to computational intractability. Yet, multiple strategies aiming to boost weight collaboration in one way or another, *e.g*. Dropout [47], WeightNorm [45], Weight Standardization [43], have proven to improve training speed, stability and generalization. Ignoring weight correlations might partially explain the shortcomings of BNNs in terms of underfitting [7]. This motivates us to find a scalable way to compute the posterior distribution of the weights without discarding their correlations and better exploit BNNs.

Li *et al*. [28] have recently found that the *intrinsic* dimension of DNNs can be in the order of hundreds to a few thousand. The good performances of BE that build on weights from a low-rank subspace further confirm this finding. For efficiency, we leverage the Rank-1 subspace decomposition in BE and estimate here the distribution of the weights, leading to a novel form of BNNs. Formally, instead of computing the posterior distribution $P(\Theta \mid \mathcal{D})$, we aim now for $P(\theta^{\text{fast}} \mid \mathcal{D})$.

A first approach would be to compute Rank-1 weight distributions by using $\mathbf{r}_j$ and $\mathbf{s}_j$ as variational layers, place priors on them and compute their posterior distributions in a similar manner to [3]. Dusenberry *et al*. [7] show that these Rank-1 BNNs stabilize training by reducing the variance of the sampled weights, due to sampling only from Rank-1 variational distributions instead of full weight matrices. However this raises the memory cost significantly as training is performed simultaneously over all J sub-networks: on CIFAR-10 for ResNet-50 with $J{=}4$ the authors use 8 TPUv2 cores with mini-batches of size 64 per core.

We argue that a more efficient way of computing the posterior distribution of the fast weights would be to learn instead the posterior distribution of the lower dimensional latent variables of $\{\mathbf{r}, \mathbf{s}\} \in \theta^{\text{fast}}$. This can be efficiently done with a VAE [25] that can find a variational approximation $Q_\phi(\mathbf{z} \mid \mathbf{r})$ to the intractable posterior $P_\psi(\mathbf{z} \mid \mathbf{r})$. VAEs can be seen as a generative model that can deal with complicated dependencies between input dimensions via a probabilistic encoder that projects the input into a latent space following a specific prior distribution. For simplicity and clarity, from here onward we derive our formalism only for $\mathbf{r}$ at a single layer and consider weights $\mathbf{s}$ to be deterministic. Here the input to the VAE are the weights $\mathbf{r}$ and we rely on it to learn the dependencies between weights and encode them into the latent representation.

In detail, for each layer of the network $f_\Theta(\cdot)$ we introduce a VAE composed of a one layer encoder $g_\phi^{\text{enc}}(\cdot)$ with variational parameters ϕ and a one layer decoder $g_\psi^{\text{dec}}(\cdot)$ with parameters ψ. Let the prior over the latent variables be a centered isotropic Gaussian $P_\psi(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, \mathbf{I})$. Like common practice, we let the variational approximate posterior $Q_\phi(\mathbf{z} \mid \mathbf{r})$ be a multivariate Gaussian with diagonal covariance. The encoder takes as input a mini-batch of size J (the size of the ensemble) composed of all the $\mathbf{r}_j$ weights of this layer and outputs as activations $(\boldsymbol{\mu}_j, \boldsymbol{\sigma}_j^2)$. We sample a latent variable $\mathbf{z}_j \sim \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\sigma}_j^2 \mathbf{I})$ and feed it to the decoder, which in turn outputs the reconstructed weights $\hat{\mathbf{r}}_j = g_\psi^{\text{dec}}(\mathbf{z}_j)$. In other words, at each forward pass, we sample new fast weights $\hat{\mathbf{r}}_j$ from the latent posterior distribution to be further used for generating the ensemble. The weights of each member of the ensemble $\overline{\mathbf{W}}_j = \mathbf{W}_{\text{share}} \odot (\hat{\mathbf{r}}_j \mathbf{s}_j^\top)$ are now random variables depending on $\mathbf{W}_{\text{share}}$, $\mathbf{s}_j$ and $\mathbf{z}_j$. Note that while in practice we sample J weight configurations, this approach allows us to generate larger ensembles by sampling multiple times from the same latent distribution. We illustrate an overview of an LP-BNN layer in Figure 3.

The VAE modules are trained in the standard manner with the ELBO loss [25] jointly with the rest of the network. The final loss function is:

$$\mathcal{L}_{\text{LP-BNN}}(\Theta^{\text{LP-BNN}}) = -\sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} \mathbb{E}_{\mathbf{z} \sim Q_\phi(\mathbf{z} \mid \mathbf{r})} \log \left(P(y_i \mid \mathbf{x}_i, \Theta^{\text{LP-BNN}}, \mathbf{z}) \right) + \text{KL}(Q_\phi(\mathbf{z} \mid \mathbf{r}) || P_\psi(\mathbf{z})) + \|\mathbf{r} - \hat{\mathbf{r}}\|^2, \quad (7)$$

where $\Theta^{\text{LP-BNN}} = \left\{ \theta^{\text{slow}}, \theta^{\text{fast}}{:}\{\mathbf{r}_j, \mathbf{s}_j\}_{j=1}^J, \theta^{\text{variational}}{:}\{\phi, \psi\} \right\}$. The loss is applied to all J members of the ensemble.

At a first glance, the loss $\mathcal{L}_{\text{LP-BNN}}$ bears some similarities with $\mathcal{L}_{\text{BNN}}$ (Eq. 5). Both losses have a likelihood and KL term. The likelihood in $\mathcal{L}_{\text{BNN}}$, *i.e*. the cross-entropy loss, depends on input data $\mathbf{x}_i$ and on the parameters Θ sampled from $Q_\nu(\Theta)$, while $\mathcal{L}_{\text{LP-BNN}}$ depends on the latent variables $\mathbf{z}_j$ sampled from $Q_\phi(\mathbf{z}_j \mid \mathbf{r}_j)$ that lead to the fast weights $\hat{\mathbf{r}_j}$. It guides the weights towards useful values for the main task. The KL term in $\mathcal{L}_{\text{BNN}}$ enforces the per-weight prior, while in $\mathcal{L}_{\text{LP-BNN}}$ it preserves the consistency and simplicity of the common latent distribution of the weights $\mathbf{r}_j$. In addition, $\mathcal{L}_{\text{LP-BNN}}$ has an input weight reconstruction loss (last term in Eq. 7) ensuring that the generated weights $\hat{\mathbf{r}}_j$ are still compatible with the rest of parameters of the network and do not cause high variance and instabilities during training, as typically occurs in standard BNNs [7].

At test time, we generate the LP-BNN ensemble on the fly by sampling the weights $\hat{\mathbf{r}}_j$ from the encodings of $\mathbf{r}_j$ to compute $\overline{\mathbf{W}}_j$. For the final prediction we compute the empirical mean of the likelihoods of the ensemble:

$$P(y_i | \mathbf{x}_i) = \frac{1}{J} \sum_{j=1}^{J} P(y_i \mid \mathbf{x}_i, \theta^{\text{slow}}, \mathbf{s}_j, \hat{\mathbf{r}}_j) \quad (8)$$

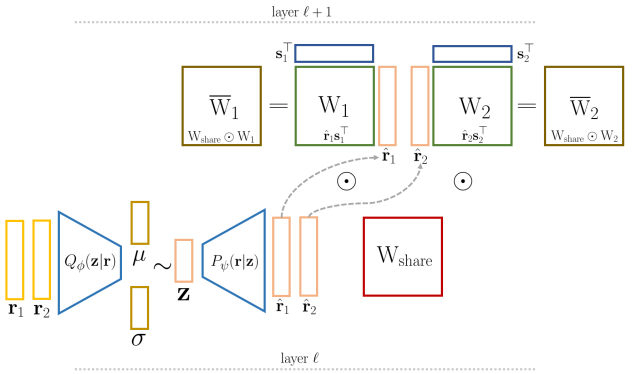

Figure 3: Illustration on how LP-BNN generates the ensemble weights ($J = 2$) using sampled fast weights $\hat{\mathbf{r}}_1$ and $\hat{\mathbf{r}}_2$.

3.2. The utility of Rank-1 Priors

One could ask why using the Rank-1 formalism, instead of simply feeding the weights a layer to the VAE to infer the latent distribution. Rank-1 prior reduce significantly the number of weights to train the VAE over, due to the decomposition of the fast weights into $\mathbf{r}$ and $\mathbf{s}$. This further allows us to consider multiple such weights, J, enabling faster training of the VAE as its training samples are more numerous and more diverse.

Next, we establish connections between the cardinality J of the ensemble and the posterior covariance matrix. Our prior distribution allows for the introduction of correlations between weights, which is a desirable property due to its superior expressiveness [7] but which can be otherwise difficult to approximate. Also, the covariance matrix of our prior is a Rank-1 matrix. Thanks to the Eckart-Young theorem [48] (Theorem 5.1), we can quantify the error of approximating the covariance by a Rank-1 matrix, based on the second up to the last singular values.

Let us denote by $\Theta_1, \ldots, \Theta_J$ the J weights trained by our algorithm, $\Theta_{\text{avg}} = 1/J \sum_{j=1}^{J} \Theta_j$ and $\Delta_j = \Theta_j - \Theta_{\text{avg}}$. The differences and the sum in the previous equations are calculated element-wise on all the weights of the DNNs. Then, for each new data sample $\mathbf{x}$, the prediction of the DNN $f_{\Theta_{\text{avg}}}(\cdot)$ is equivalent to the average of the DNNs $f_{\Theta_j}(\cdot)$ applied on $\mathbf{x}$:

$$\frac{1}{J}\sum_{j=1}^{J} f_{\Theta_j}(\mathbf{x}) = f_{\Theta_{\text{avg}}}(\mathbf{x}) + \mathcal{O}\left(\|\Delta\|^2\right) \tag{9}$$

with $\|\Delta\| = \max_j \|\Delta_j\|$ where the L_2 is computed over all weights. The proof can be found in Section 3.5 of [23]. It follows that in fact we do not learn a Rank-1, but an up to Rank-J covariance matrix, if all the $\mathbf{s}_j$ $\mathbf{r}_j$ are independent. Hence the choice of J acts as an approximation factor of the covariance matrix. Wen *et al*. [50] tested different values of J and found that $J = 4$ was the best compromise, which we will also use here.

3.3. Computational complexity

Recent efforts [9, 51] studied the weight modes computed by Deep Ensembles within BNNs, yet this line of research is computationally intractable at the scale required for practical computer vision tasks. Dusenberry *et al*. [7] propose a more scalable solution for image classification, which is nonetheless prone to high instabilities due to the important number of parameters and to the fact that $\mathbf{r}_j$ and $\mathbf{s}_j$ are the latent variables of the variational distribution. In comparison, our approach requires less memory resources since we encode $\mathbf{r}_j$ in a lower dimensional space (we found empirically that a latent space of size only 32 provides an appealing compromise between accuracy and compactness). The only additional cost in terms of parameters and memory used w.r.t. BE is related to the compact VAEs associated with each layer.

Besides the lower number of parameters, LP-BNN training is more stable than for Rank-1 BNN due to the reconstruction term $\|\mathbf{r}_j - \hat{\mathbf{r}}_j\|_2^2$ which regularizes the $\mathcal{L}_{\text{LP-BNN}}$ loss in Eq. (7) by controlling the variances of the sampled weights. BNNs usually need a range of heuristics, e.g. clipping, initialization from truncated Normal, extra weight regularization to stabilize training [7]. For LP-BNN training is overall straightforward even on complex and deep models, *e.g*. DeepLabV3+, thanks to the VAE module that is stable.

3.4. Discussion on uncertainty with LP-BNN

The predictive uncertainty of a DNN stems from two main types of uncertainty [21]: *aleatoric uncertainty* and *epistemic uncertainty*. The former is related to randomness, typically due to the noise in the data. The latter concerns finite size training datasets. The epistemic uncertainty captures the uncertainty in the DNN parameters and their lack of knowledge on the model that generated the training data.

In BNN approaches, through likelihood marginalization over weights, the prediction is computed by integrating the outputs from different DNNs weighted by the posterior distribution (Eq. 3) allowing to conveniently capture both types of uncertainties [34]. The quality of the uncertainty estimates depends on the diversity of predictions and views provided by the BNN. DE [27] achieve excellent diversity [9] by mimicking BNN ensembles through training of multiple individual models. Recently, Wilson *et al*. [51] propose to combine DE and BNN towards improving diversity further. However, as DE are already computationally demanding, we argue that BE is a more pragmatic choice for increasing the diversity of our BNN , leading to better uncertainty quantification.

Figure 5 shows comparative results on diversity provided by different DNNs. We train LP-BNN, BE and DE with WideResnet-28-10 [55] on CIFAR-10 [26]. We evaluate them on the test sets of CIFAR-10, CIFAR10-C [18] and SVHN [39]. The SVHN images (representing digits) are different from the training data, *i.e*. predominant epistemic uncertainty, while CIFAR10-C data are different from the

training images due to noise corruption, *i.e*. more aleatoric uncertainty. The first row of Figure 5 shows the test images. The next three rows synthesize diversity results obtained with LP-BNN, BE and DE respectively. For all methods we set the number of models to $J = 4$. The expected behavior is that different DNNs would not predict the same class on the OOD images, reducing the confidence score of the DNN. We can see that the diversity of BE is inferior for CIFAR10-C and SVHN, leading to poor results in Table 2.

4. Related work

Bayesian methods and deep learning. Bayesian approaches have an important place in the machine learning community. MacKay [30, 32] was among the first to establish links between the two domains, then Neal [38] showed that considering a neural network composed of one layer with an infinite number of neurons was equivalent to a Gaussian process regression. In this light, the use of some fundamental regularization techniques such as weight decay comes down as considering a Bayesian prior on the data.

Bayesian Neural Networks. Modern BNNs are different from previous Bayesian approaches. Most of the contemporary approaches [3, 7, 13, 29, 46, 53] consider that the weights of the DNNs follow a specific distribution. Given a prior over the weights, their goal is to estimate the posterior distribution. Gaussian priors [3] were traditionally considered due to the link to weight decay. In [36, 46], studies over various prior distributions were performed. In [7] were used. More advanced prior distributions were proposed in [13, 29, 53]. In [13] the prior distribution is a horseshoe prior [4] which promotes sparsity. Here we propose a Rank-1 prior over the latent space leading to non-independent weights.

Ensemble Approaches. Lakshminarayanan *et al*. [27] proposed an effective non-Bayesian approach consisting in training an ensemble of DNNs leading to top uncertainty quantification performance. However, DE scale linearly in both training and testing with the size of the ensemble. SWAG [33] and TRADI [11] are methods to track the posterior distribution of the DNN in a more efficient manner , and then at test time to ensemble the prediction from various sample DNNs. OVNNI [10] and OVADM [40] ensemble the results of multiple *One vs. All* trainings, leading to high performance to detect anomalies. Franchi *et al*. [11] and Mehrtash *et al*. [35] construct an ensemble from random perturbations of the weights of a trained network. Our prior distribution over the weights of the DNN is a multivariate normal distribution, in contrast to previous works that consider independent Gaussian distributions.

5. Experiments and results

5.1. Implementation details

We evaluate the performance of LP-BNN in assessing the uncertainty of its predictions. For our benchmark, we evaluated LP-BNNon different scenarios against several related baselines with different advantages in terms of performance, training or runtime: BE, DE, Maximum Class Probability (MCP), MC Dropout [12], TRADI [11].

First, we evaluate the predictive performance in terms of accuracy for image classification and mIoU [8] for semantic segmentation respectively. Secondly, we evaluate the quality of the confidence scores provided by the DNNs by means of Expected Calibration Error (ECE) [15]. For ECE we use M-bin histograms of confidence scores and accuracy, and compute the average of M bin-to-bin differences between the two histograms. Similarly to [15] we set $M = 15$. To evaluate the DNNs on corrupted images and dataset shift, we first train the DNNs on CIFAR-10 [26] or CIFAR-100 [26] and then test on the corrupted versions of these datasets [18]. The corruptions include different types of noise, blurring, and some other transformations that alter the quality of the images. For this scenario, similarly to [49], we use as evaluation measures the Corrupted Accuracy (cA) and Corrupted Expected Calibration Error (cE), that offer a better understanding of the behavior of our DNN when facing aleatoric uncertainty.

In order to evaluate the epistemic uncertainty, we propose to assess the OOD detection performance. This scenario typically consists in training a DNN over a dataset following a given distribution, and testing it on data coming from this distribution and data from another distribution. We would quantify the confidence of the DNN predictions in this setting by using theirs prediction scores, *i.e*. output softmax values. We use the same indicators of the accuracy of detecting OOD data as in [19]: AUC, AUPR, and the FPR-95%-TPR. These metrics measure whether the DNN model lacks knowledge regarding some specific data and how reliable are its predictions.

5.2. Image classification with CIFAR-10/100 [26]

Protocol. Here we train on CIFAR-10 [26] composed of 10 classes. For CIFAR-10 we consider as OOD the SVHN dataset [39]. Since SVHN is a color image dataset of digits, it guarantees that the OOD data comes from a distribution different from those of CIFAR-10. We use WideResNet-28-10 [55] for all methods, a popular architecture for this dataset and evaluate on CIFAR-10-C [18]. For CIFAR-100 [26] we use again WideResNet-28-10 [55] and test on the test set of CIFAR-100 and of CIFAR-100-C [18]. Note that for all DNNs, even for DE, results are averaged over 3 random seeds, for statistical relevance.We use cutout [6] as data augmentation, as commonly used for these datasets.

Method	CIFAR-10 Acc ↑	AUC ↑	AUPR ↑	FPR-95-TPR ↓	ECE ↓	cA ↑	cE ↓	CIFAR-100 Acc ↑	ECE ↓	cA ↑	cE ↓
MCP + cutout	96.33	0.9600	0.9767	0.115	0.0207	32.98	0.6167	80.19	0.1228	19.33	0.7844
MC dropout	95.95	0.9126	0.9511	0.282	0.0172	32.32	0.6673	75.40	0.0694	19.33	0.5830
MC dropout +cutout	96.50	0.9273	0.9603	0.242	0.0117	32.35	0.6403	77.92	0.0572	27.66	0.5909
Deep Ensembles + cutout	**96.74**	**0.9803**	**0.9896**	**0.071**	**0.0093**	68.75	0.1414	**82.29**	**0.0524**	47.35	**0.1981**
BatchEnsembles + cutout	96.48	0.9540	0.9731	0.132	0.0167	**71.67**	0.1928	81.27	0.0912	47.44	0.2909
LP-BNN (ours) + cutout	95.02	**0.9691**	**0.9836**	**0.103**	**0.0094**	**69.51**	**0.1197**	76.85	0.0677	**47.80**	0.2324

Table 1: Comparative results for classification tasks on CIFAR-10 and CIFAR-100. The results are averaged over three seeds.

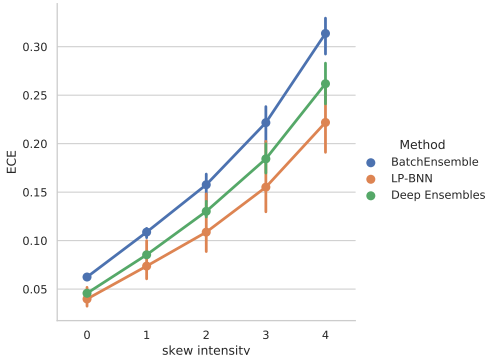

Figure 4: Study of the evolution of the ECE of LP-BNN, Deep Ensembles and BatchEnsemble on CIFAR-10-C with different levels of corruption.

Please find in the supplementary the hyperparameters for this experiment.

Discussion. We illustrate results for this experiments in Table 2. We notice that DE with cutout outperforms others on most of the metrics except ECE, cA, and cE on CIFAR-10, and cA on CIFAR-100, where LP-BNN achieves state of the art results. This means that LP-BNN is competitive for aleatoric uncertainty estimation. In fact, ECE is calculated on the test set of CIFAR-10 and CIFAR-100, so it mostly measures the reliability of the confidence score in the training distribution. cA and cE are evaluated on corrupted versions of CIFAR-10 and CIFAR-100, which amounts to quantifying the aleatoric uncertainty. We can see that for this kind of uncertainty, LP-BNN achieves state of the art performance. On the other hand, for epistemic uncertainty, we can see that DE always attain best results. Yet, LP-BNN, in most cases, performs close to DE. Computation wise, DE takes 52 hours to train on CIFAR-10, while our solution needs 2 times less, 26 hours and 30 minutes. Overall, our LP-BNN is more computationally efficient while providing better results for the aleatoric uncertainty. As a comparison, it takes 26 hours for BE to be train on CIFAR-10, sensibly less than LP-BNN. In Figure 4 and Table 3 we observe that our method exhibits the best global ECE on CIFAR-10-C, as well as the best ECE for the stronger corruptions.

5.3. Semantic segmentation

We evaluate next semantic segmentation, a task of interest for autonomous driving, for which high capacity DNNs are used for processing high resolution images with complex urban scenery.

StreetHazards [17]. StreetHazards is a large-scale dataset that consists of different sets of synthetic images of street scenes. More precisely, this dataset is composed of 5125 images for training and 1500 test images. The training dataset contains pixel-wise annotations for 13 classes. The test dataset comprises 13 training classes and 250 OOD classes, unseen during in the training set, making it possible to test robustness of the algorithm when facing a diversity of possible scenarios. For this experiment, we used DeepLabv3+ [5] with a ResNet-50 encoder [16]. Following the implementation in [17], most papers use PSPNet [56] that aggregates predictions over multiple scales, an ensembling that can obfuscate in the evaluation the uncertainty contribution of a method. This can partially explain the excellent performance of MCP on the original settings [17]. We propose using DeepLabv3+ instead as it enables a clearer evaluation of the predictive uncertainty. We propose two DeepLabv3+ variants as follows. DeepLabv3+ is composed of an encoder network and decoder network; in the first version, we change the decoder by replacing all the convolutions with our new version of LP-BNN convolutions and leave the encoder unchanged. In the second variant we use weight standardization [42] on the convolutional layers of the decoder, replacing batch normalization [22] in the decoder with group normalization [52]. We denote the first version LP-BNN and the second one LP-BNN + GN.

BDD-Anomaly [17]. BDD-Anomaly is a subset of the BDD100K dataset [54], composed of 6688 street scenes for training and 361 for the test set. The training set contains pixel-level annotations for 17 classes, and the test dataset is composed of the 17 training classes and 2 OOD classes: motor-cycle and train. For this experiment, we use DeepLabv3+[5] with the experimental protocol from [17]. As previously we use ResNet50 encoder [16]. For this experiment, we use the LP-BNN and LP-BNN + GN variants.

Discussion. We emphasize that the semantic segmentation is more challenging than the CIFAR classification since images

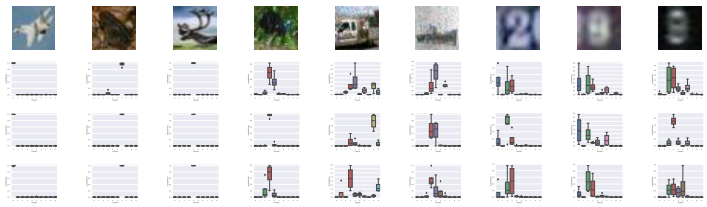

Figure 5: Illustration of the diversity provided by the different ensembles. The first row contains in order three images from the test set of CIFAR-10, of CIFAR-10-C and of SVHN. The next three rows represent the corresponding outputs of the different sub models for the three ensambling algorithms being considered: LP-BNN, BatchEnsemble and Deep Ensembles.

Dataset	OOD method	mIoU ↑	AUC ↑	AUPR ↑	FPR-95-TPR ↓	ECE ↓
StreetHazards DeepLabv3+ ResNet50	Baseline (MCP)	53.90	0.8660	0.0691	0.3574	0.0652
	TRADI	52.46	0.8739	0.0693	0.3826	0.0633
	Deep Ensembles	55.59	0.8794	**0.0832**	0.3029	0.0533
	BatchEnsemble	**56.16**	0.8817	0.0759	0.3285	0.0609
	LP-BNN (ours)	54.50	**0.8833**	0.0718	0.3261	**0.0520**
	LP-BNN + GN (ours)	**56.12**	**0.8908**	0.0742	**0.2999**	**0.0593**
BDD-Anomaly DeepLabv3+ ResNet50	Baseline (MCP)	47.63	0.8515	0.0450	0.2878	0.1768
	TRADI	44.26	0.8480	0.0454	0.3687	0.1661
	Deep Ensembles	**51.07**	0.8480	0.0524	**0.2855**	**0.1419**
	BatchEnsemble	48.09	0.8427	0.0449	0.3017	0.1690
	LP-BNN (ours)	49.01	**0.8532**	0.0452	0.2947	0.1716
	LP-BNN + GN (ours)	47.15	**0.8553**	**0.0577**	0.2866	0.1623

Table 2: Comparative results obtained on the OOD task for semantic segmentation. The results are averaged over three seeds.

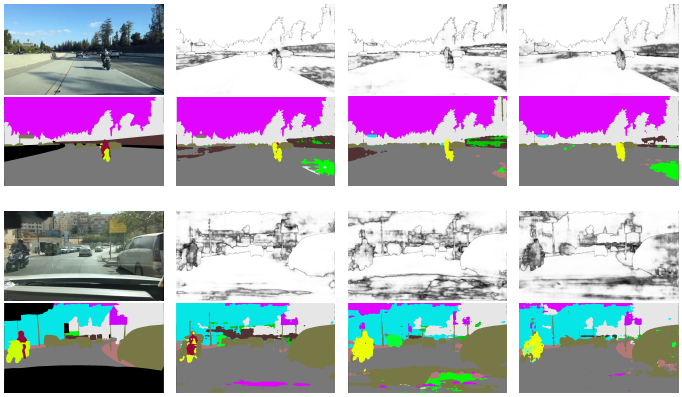

Figure 6: Visual assessment on two images of BDD-Anomaly in which a motorcycle (OOD class) is present. For each image: on the first row - input image and confidence maps for MCP, BE and LP-BNN (ours); on the second row - GT segmentation and segmentation maps for MCP, BE and LP-BNN (ours). LP-BNN is less confident on the OOD objects.

are bigger, the content is more complex.The larger input size constrains to the use of a smaller batches. This is crucial since the fast weights of the ensemble layers are trained just on one batch slice. In this experiment, we could use batches of size 4 and train the fast weights on slices of size 1. Yet, despite these computational difficulties, with our technique, we achieved state-of-the-art results for most metrics. We can see in Table 2 that our strategies achieve state-of-the-art performance in detecting OOD data and are well calibrated. We can also see in Figure 6, where the OOD class is the motorcycle, that our DNN is less confident in this class. Hence LP-BNN allows us to have a more reliable DNN which is essential for real-world applications.

6. Conclusion

We propose a new BNN framework able to quantify uncertainty in the context of deep learning. Owing to each layer of the network being tied to and regularized by a VAE, LP-BNNs are stable, efficient, and therefore easy to train compared to existing BNN models. The extensive empirical comparisons on multiple tasks show that LP-BNNs reach state-of-the-art levels with substantially lower computational cost. We hope that our work will open new research paths on effective training of BNNs. In the future we intend to explore new strategies for plugging more sophisticated VAEs in our models along with more in-depth theoretical studies.

References

[1] Arsenii Ashukha, Alexander Lyzhov, Dmitry Molchanov, and Dmitry Vetrov. Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. In *International Conference on Learning Representations*, 2020. 4321, 4323

[2] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. MIT press Massachusetts, USA:, 2017. 4322

[3] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *International Conference on International Conference on Machine Learning*, page 1613–1622, 2015. 4321, 4323, 4324, 4326, 2

[4] Carlos M Carvalho, Nicholas G Polson, and James G Scott. Handling sparsity via the horseshoe. In *Artificial Intelligence and Statistics*, pages 73–80, 2009. 4326

[5] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018. 4327

[6] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 4326

[7] Michael W. Dusenberry, Ghassen Jerfel, Yeming Wen, Yian Ma, Jasper Snoek, Katherine Heller, Balaji Lakshminarayanan, and Dustin Tran. Efficient and scalable bayesian neural nets with rank-1 factors. In *International Conference on Machine Learning (ICML)*, 2020. 4322, 4323, 4324, 4325, 4326

[8] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015. 4326

[9] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*, 2019. 4321, 4323, 4325

[10] Gianni Franchi, Andrei Bursuc, Emanuel Aldea, Severine Dubuisson, and Isabelle Bloch. One versus all for deep neural network incertitude (ovnni) quantification. *arXiv preprint arXiv:2006.00954*, 2020. 4326

[11] Gianni Franchi, Andrei Bursuc, Emanuel Aldea, Séverine Dubuisson, and Isabelle Bloch. Tradi: Tracking deep neural network weight distributions. In *Proceedings of the European conference on computer vision (ECCV)*, 2020. 4321, 4322, 4323, 4326

[12] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016. 4326

[13] Soumya Ghosh, Jiayu Yao, and Finale Doshi-Velez. Structured variational learning of bayesian neural networks with horseshoe priors. In *International Conference on Machine Learning*, pages 1744–1753, 2018. 4326

[14] Alex Graves. Practical variational inference for neural networks. In *Advances in neural information processing systems*, pages 2348–2356, 2011. 4323

[15] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330. JMLR. org, 2017. 4326

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 4324, 4327

[17] Dan Hendrycks, Steven Basart, Mantas Mazeika, Mohammadreza Mostajabi, Jacob Steinhardt, and Dawn Song. A benchmark for anomaly segmentation. *arXiv preprint arXiv:1911.11132*, 2019. 4327, 1

[18] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2018. 4325, 4326

[19] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017*, 2017. 4326

[20] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13, 1993. 4322

[21] Stephen C Hora. Aleatory and epistemic uncertainty in probability elicitation with an example from hazardous waste management. *Reliability Engineering & System Safety*, 54(2-3):217–223, 1996. 4325

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 4327

[23] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, pages 876–885, 2018. 4325

[24] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999. 4322

[25] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. 4322, 4323, 4324

[26] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 4325, 4326, 1

[27] 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. 4321, 4322, 4323, 4325, 4326

[28] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective

landscapes. In *International Conference on Learning Representations*, 2018. 4324

[29] Christos Louizos, Xiahan Shi, Klamer Schutte, and Max Welling. The functional neural process. In *Advances in Neural Information Processing Systems*, pages 8746–8757, 2019. 4326

[30] David JC MacKay. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992. 4326

[31] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992. 4322

[32] David JC MacKay. Probable networks and plausible predictions—a review of practical bayesian methods for supervised neural networks. *Network: computation in neural systems*, 6(3):469–505, 1995. 4326

[33] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. In *Advances in Neural Information Processing Systems*, volume 32, pages 13153–13164, 2019. 4321, 4322, 4323, 4326

[34] Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks. In *Advances in Neural Information Processing Systems*, pages 7047–7058, 2018. 4325

[35] Alireza Mehrtash, Purang Abolmaesumi, Polina Golland, Tina Kapur, Demian Wassermann, and William Wells. Pep: Parameter ensembling by perturbation. *Advances in Neural Information Processing Systems*, 33, 2020. 4323, 4326

[36] Eric Thomas Nalisnick. *On priors for bayesian neural networks*. PhD thesis, UC Irvine, 2018. 4326

[37] Radford M Neal. Bayesian learning for neural networks. *PhD thesis, University of Toronto*, 1995. 4322

[38] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012. 4326

[39] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011. 4325, 4326

[40] Shreyas Padhy, Zachary Nado, Jie Ren, Jeremiah Liu, Jasper Snoek, and Balaji Lakshminarayanan. Revisiting one-vs-all classifiers for predictive uncertainty and out-of-distribution detection in neural networks. *arXiv preprint arXiv:2007.05134*, 2020. 4326

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035, 2019. 1

[42] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Rethinking normalization and elimination singularity in neural networks. *arXiv preprint arXiv:1911.09738*, 2019. 4327

[43] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Weight standardization. *arXiv preprint arXiv:1903.10520*, 2019. 4323, 4324

[44] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on International Conference on Machine Learning-Volume 32*, pages II–1278, 2014. 4323

[45] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016. 4322, 4323, 4324

[46] Daniele Silvestro and Tobias Andermann. Prior choice affects ability of bayesian neural networks to identify unknowns. *arXiv preprint arXiv:2005.04987*, 2020. 4326

[47] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014. 4322, 4323, 4324

[48] Jianzhong Wang. *Geometric structure of high-dimensional data and dimensionality reduction*. Springer, 2012. 4325

[49] Yeming Wen, Ghassen Jerfel, Rafael Muller, Michael W Dusenberry, Jasper Snoek, Balaji Lakshminarayanan, and Dustin Tran. Improving calibration of batchensemble with data augmentation. In *ICML 2020 workshop on Uncertainty Robustness in Deep Learning*, 2020. 4326

[50] Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In *International Conference on Learning Representations*, 2020. 4323, 4325

[51] Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. In *Advances in Neural Information Processing Systems*, 2020. 4325

[52] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018. 4327

[53] Wanqian Yang, Lars Lorch, Moritz A Graule, Srivatsan Srinivasan, Anirudh Suresh, Jiayu Yao, Melanie F Pradier, and Finale Doshi-Velez. Output-constrained bayesian neural networks. In *2019 ICML Workshop on Uncertainty and Robustness in Deep Learning (UDL)*, 2019. 4326

[54] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2636–2645, 2020. 4327

[55] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12, September 2016. 4325, 4326

[56] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017. 4327

Encoding the latent posterior of Bayesian Neural Networks for uncertainty quantification (Supplementary material)

Hyper-parameter	StreetHazards	BDD-Anomaly
Ensemble size J	4	4
learning rate	0.1	0.1
batch size	4	4
number of train epochs	25	25
weight decay for θ^{slow} weights	0.0001	0.0001
weight decay for θ^{fast} weights	0.0	0.0
cutout	True	True
SyncEnsemble BN	False	False
Group Normalisation	True	True
Size of the latent space $\mathbf{z}$	32	32

Table 1: Values of the hyper-parameters used in the semantic segmentation experiments (§5.3).

7. Implementation details

For our implementation, we use PyTorch [41] and will release the code after the review in order to facilitate reproducibility and further progress. In the following we share the hyper-parameters for our experiments on image classification and semantic segmentation.

7.1. Semantic segmentation experiments

In Table 1, we summarize the hyper-parameters used in the StreetHazards [17] and BDD-Anomaly [17] experiments.

7.2. Image classification experiments

In Table 2, we summarize the hyper-parameters used in the CIFAR-10 [26] and CIFAR-100 [26] experiments.

8. Notations

In Table 3, we summarize the main notations used in the paper. Table 3 should facilitate the understanding of §2 (the preliminaries) and of §3 (the presentation of our approach).

Hyper-parameter	CIFAR-10	CIFAR-100
Ensemble size J	4	4
initial learning rate	0.1	0.1
batch size	128	128
lr decay ratio	0.1	0.1
lr decay epochs	[80, 160, 200]	[80, 160, 200]
number of train epochs	250	250
weight decay for θ^{slow} weights	0.0001	0.0001
weight decay for θ^{fast} weights	0.0	0.0
cutout	True	True
SyncEnsemble BN	False	False
Size of the latent space $\mathbf{z}$	32	32

Table 2: Values of the hyper-parameters used in the classification experiments (§5.2).

Notations	Meaning
$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$	the set of n data samples and the corresponding labels
Θ	the set of weights of a DNN
$P(\Theta)$	the prior distribution over the weights of a DNN
$Q_\nu(\Theta)$	the variational prior distribution over the weights of a DNN used in standard BNNs [3]
ν	the parameters of the variational prior distribution over the weights of a DNN used in standard BNNs [3]
$P(y_i \mid \mathbf{x}_i, \Theta)$	the likelihood that DNN outputs y_i following a prediction over input image $\mathbf{x}_i$
J	the number of ensembling DNNs
$\theta^{\text{slow}} = \{\text{W}_{\text{share}}\}$	the shared "slow" weights of the network
$\theta^{\text{fast}} = \{\text{W}_j\}_{j=1}^J = \{(\mathbf{r}_j, \mathbf{s}_j)\}_{j=1}^J$	the set of individual "fast" weights of BatchEnsemble for ensembling of J networks
$\theta^{\text{fast}} = \{(\hat{\mathbf{r}}_j, \mathbf{s}_j)\}_{j=1}^J$	the set of fast weights of LP-BNN for ensembling of J networks. $\hat{\mathbf{r}}_j$ are sampled from the latent weight space of weights $\mathbf{r}_j$.
$\theta^{\text{variational}} = \{(\phi_j, \psi_j)\}_{j=1}^J$	the parameters of the VAE for computing the low dimensional latent distribution of $\mathbf{r}_j$
$g_\phi^{\text{enc}}(\cdot)$	the encoder of the VAE applied on $\mathbf{r}$
$g_\psi^{\text{dec}}(\cdot)$	the decoder of the VAE for reconstructing $\hat{\mathbf{r}}$ from latent code of $\mathbf{r}$
$Q_\phi(\mathbf{z} \mid \mathbf{r})$	the variational distribution over the weights $\mathbf{r}$ to approximate the intractable posterior $P_\psi(\mathbf{z} \mid \mathbf{r})$
$(\boldsymbol{\mu}_j, \boldsymbol{\sigma}_j) = g_\phi^{\text{enc}}(\mathbf{r}_j)$	encoder output that parameterize a multivariate Gaussian with diagonal covariance
$\mathbf{z}_j \sim Q_\phi(\mathbf{z} \mid \mathbf{r}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_j, \boldsymbol{\sigma}_j^2 \mathbf{I})$	sampling a latent code $\mathbf{z}$ from the latent distribution
$P_\psi(\mathbf{z}_j) = \mathcal{N}(\mathbf{z}; 0, \mathbf{I})$ with $j \in [1, J]$	the prior distribution on $\mathbf{z}_j$
$\hat{\mathbf{r}}_j = g_\psi^{\text{dec}}(\mathbf{z}_j)$	the reconstruction of $\mathbf{r}_j$ from its latent distribution , *i.e.* the variational fast weights
$\overline{\text{W}}_j = \text{W}_{\text{share}} \odot (\mathbf{r}_j \mathbf{s}_j^\top)$	the weight of a BatchEnsemble network j computed from slow and fast weights where $\odot$ is the Hadamard product and $(\mathbf{r}_j \mathbf{s}_j^\top)$ the inner product between these two vectors.
$\overline{\text{W}}_j = \text{W}_{\text{share}} \odot (\hat{\mathbf{r}}_j \mathbf{s}_j^\top)$	the weight of LP-BNN network network j computed from slow and variational fast weights

Table 3: Summary of the main notations of the paper.