Stochastic Weight Sharing for Bayesian Neural Networks

Moule Lin
Lero, Trinity College Dublin

Shuhao Guan
University College Dublin

Weipeng Jing
Northeast Forestry University

Goetz Botterweck
Lero, Trinity College Dublin

Andrea Patane
Lero, Trinity College Dublin

Abstract

While offering a principled framework for uncertainty quantification in deep learning, the employment of Bayesian Neural Networks (BNNs) is still constrained by their increased computational requirements and the convergence difficulties when training very deep, state-of-the-art architectures. In this work, we reinterpret weight-sharing quantization techniques from a stochastic perspective in the context of training and inference with Bayesian Neural Networks (BNNs). Specifically, we leverage 2D-adaptive Gaussian distributions, Wasserstein distance estimations, and alpha-blending to encode the stochastic behaviour of a BNN in a lower-dimensional, soft Gaussian representation. Through extensive empirical investigation, we demonstrate that our approach significantly reduces the computational overhead inherent in Bayesian learning by several orders of magnitude, enabling the efficient Bayesian training of large-scale models, such as ResNet-101 and Vision Transformer (VIT). On various computer vision benchmarks—including CIFAR-10, CIFAR-100, and ImageNet1k—our approach compresses model parameters by approximately 50× and reduces model size by 75%, while achieving accuracy and uncertainty estimations comparable to the state-of-the-art.

1 INTRODUCTION

Bayesian Neural Networks (BNNs) promise to combine the representational capacity of deep learning with principled uncertainty estimations enabled by means of Bayesian learning theory (Hinton and Neal, 1995). Arguably, this combination makes them particularly appealing for safety-critical machine learning applications where the quantification of uncertainty is of paramount importance (Forsberg et al., 2020). Indeed, they have been widely employed in scenarios like e-Health (Marcos et al., 2010), robust control (Wicker et al., 2024), autonomous driving (Michelmore et al., 2020), human-in-the-loop applications (Treiss et al., 2021), automated diagnosis (Billah and Javed, 2022) and many others (Lampinen and Vehtari, 2001; Bharadiya, 2023; Vehtari and Lampinen, 1999).

Unfortunately, though, the principled treatment of uncertainty comes at the price of an increased pressure on computational resources, including the model size (×2 in the common case of mean-field Variational Inference (Blundell et al., 2015)), and the inference time, increased by an order of magnitude as multiple forward passes are needed (Hinton and Neal, 1995). Therefore, despite their potential, the use of BNNs in edge-AI and resource-constrained applications is still very limited (Bonnet et al., 2023). While recent works have investigated the development of techniques to tackle the aforementioned challenges, these are generally limited to the application of methods originally developed for deterministic neural networks (NNs) (Ferianc et al., 2021; Park et al., 2021; Chien and Chang, 2023), or the usage of the Bayesian paradigm at training time for model-order reduction purposes but without the uncertainty estimation at inference time (Van Baalen et al., 2020; Guo, 2018; Perrin et al., 2024; Subia-Waud and Dasmahapatra, 2024).

In this work, we present a quantisation technique specifically tailored to capture the stochastic behaviour of BNNs. More specifically, we design a stochastic weight-sharing quantisation method, called 2DGBNN, based on dynamically adaptive mini-batch 2D Gaussian Mixture Models and predicated on optimising weight distributions through metrics derived from Wasserstein distances (Chizat et al., 2020; De Palma et al., 2021), network gradients, and intra-class variance. Our technique works by reinterpreting standard weight-sharing (Subia-Waud and Dasmahapatra, 2024) from a 2D perspective, accounting for both the mean and variance of BNN's parameters in the case of mean-field Variational Inference (VI), and by giving it a stochastic semantic. At each training step, the current sum-total of

Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

network parameters is clustered using standard parameter-free techniques, and a mini-batch approach is used for the estimation of Gaussian distributions on a parameter space accounting for (possibly) millions of parameters. Representatives for each cluster are then selected, and alpha-blending techniques are used for sampling parameter realisations during forward passes through the network architecture. Thanks to its simplicity, the method can be seamlessly integrated into commonly used Bayesian approximation techniques based on Variational Inference (Blundell et al., 2015; Gal and Ghahramani, 2016; Minka, 2001; Welling and Teh, 2011).

We perform an extensive empirical investigation on the effectiveness of our method in training large-scale models and in reducing the computational footprint of BNNs. We utilize four widely used image classification datasets (i.e., MNIST (LeCun et al., 1998), CIFAR-10 (Krizhevsky, 2009), CIFAR-100 (Krizhevsky, 2009) and ImageNet1K (Deng et al., 2009)) and test the model results in four widely employed neural network architectures, including ResNet-18, ResNet-50, ResNet-101 (He et al., 2016) and Vision Transformer (ViT) (Dosovitskiy et al., 2021). Through our approach, we can reduce the number of trainable parameters in the network by up to 50×, while obtaining accuracy and uncertainty metrics comparable to the state-of-the-art.[1]

This paper makes the following main contributions:

- We introduce a stochastic weight-sharing technique specifically-tailored to BNNs and that employs Wasserstein distance, gradients, and within-class variance in order to improve model efficiency.
- We empirically demonstrate that our stochastic weight-sharing method compares favourably against quantisation methods employed for BNNs in terms of further reducing the computational requirements and better preserving accuracy and uncertainty metrics.
- In a variety of architectures, including ResNet-18, ResNet-50, ResNet-10 and ViT, we show how our technique can reduce model size by up to just a quarter of the original size, while working on par with state-of-the-art techniques for large-scale approximate BNN training.

2 RELATED WORK

Computational efficiency is one of the long-standing issues concerning the application of BNNs to edge-AI and embedded systems (Bonnet et al., 2023). Indeed, there is a vast section of the literature that aims to tackle the issue from a variety of different angles. One such closely related area is that of quantisation of the BNN's parameters (Guo, 2018), where the precision of the latter is reduced to minimise their computational footprint. Works have adapted techniques initially developed for deterministic NNs (Ferianc et al., 2021; Subedar et al., 2021; Lin et al., 2023; Dong et al., 2022; Ullrich et al., 2017) or developed new techniques that take into account the distributional behaviour of BNNs' parameters (Chien and Chang, 2023; Park et al., 2021; Yang et al., 2020a). While these techniques increase the computational efficiency of a given BNN architecture, by developing a weight-sharing technique (and therefore not only quantising but also in effect reducing the number of BNN parameters) the method we introduce is able to match their behaviour in standard BNN benchmarks, while at the same time, it allows for training of large scale models (Hernández-Lobato and Adams, 2015).

Several works have looked at reducing the number of parameters in BNNs, either by applying pruning techniques (Sharma and Jennings, 2021; Beckers et al., 2023; Roth and Pernkopf, 2018) or using low-rank approximations (Doan et al., 2024; Dusenberry et al., 2020; Swiatkowski et al., 2020). The latter work by reparameterising BNN weights and biases using a lower rank representation, enabling them to scale BNN inference to large models such as ResNet-50 (Dusenberry et al., 2020) and ViT (Doan et al., 2024) architectures, and are as such closely related to our work in that a smaller common representation is found for BNN parameters. The number of final parameters is still though generally higher than their full deterministic counterpart, and therefore cannot be used for edge-AI applications. In Section 5, we will observe that, albeit at the price of a small reduction in accuracy, our method reduces the number of parameters of 3 orders of magnitude.

Finally, a number of works have looked into applying Bayesian techniques for quantisation of deterministic NNs (Subia-Waud and Dasmahapatra, 2024; Louizos et al., 2017; Van Baalen et al., 2020; Perrin et al., 2024; Achterhold et al., 2018; Soudry et al., 2014; Yang et al., 2020b), including the application of weight-sharing techniques (Roth and Pernkopf, 2018; Subia-Waud and Dasmahapatra, 2024; Nowlan and Hinton, 2018). While these techniques provide encouraging results on the suitability of Bayesian theory for increasing the efficiency of deep learning, being specifically tailored to deterministic neural networks they cannot be applied to BNNs.

3 BAYESIAN NEURAL NETWORKS

We consider a neural network architecture $f^{\mathbf{w}} : \mathbb{R}^n \to \mathbb{R}^m$ parameterised by a vector of weights and biases $\mathbf{w} \in \mathbb{R}^{n_w}$, which we refer to collectively as parameters of the neural network. Bayesian learning of neural networks begins by placing a prior distribution, $p(\mathbf{w})$, over the networks' param-

[1] To support reproducibility, our code is available at https://github.com/moulelin/2DGBNN

eter. This is often assumed to be encoded through a vector of independent Gaussian distributions, one for each weight and bias in the BNN (Blundell et al., 2015). This prior belief is then updated given a dataset's evidence through the application of the Bayesian learning rule. Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n_\mathcal{D}}$ denote the full training dataset, $\mathbf{X} = (x_1, \dots, x_{n_\mathcal{D}})$ the combined vector of training inputs and $\mathbf{y} = (y_1, \dots, y_{n_\mathcal{D}})$ their corresponding outputs, then the posterior distribution on the weight is computed as:

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w})\, p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X})}, \tag{1}$$

where $p(\mathbf{y}|\mathbf{X}, \mathbf{w})$ is the likelihood and $p(\mathbf{y}|\mathbf{X})$ is the model evidence. Finally, given a test point x^*, the BNN's posterior predictive distribution on x^* is defined by:

$$p(y^*|x^*, \mathbf{X}, \mathbf{y}) = \int p(y^*|x^*, \mathbf{w}) p(\mathbf{w}|\mathbf{X}, \mathbf{y}) d\mathbf{w}. \tag{2}$$

Unfortunately, neither Equation (1) nor Equation (2) can generally be computed exactly (Hinton and Neal, 1995). Therefore a variety of approximate Bayesian inference techniques have been developed in the literature, with the two most prominent classes of approaches being based on Monte Carlo algorithms (Hinton and Neal, 1995) or on Variational Inference methods (Blundell et al., 2015). While the former provides the gold standard in terms of approximation accuracy in small architectures, Variational Inference (VI) guarantees better scalability and is therefore the focus of this paper. Briefly, VI works by approximating the true posterior, $p(\mathbf{w}|\mathbf{X}, \mathbf{y})$, by optimising the KL divergence over a simpler, parameterised distribution, $q(\mathbf{w})$, often a multidimensional Gaussian distribution with diagonal covariance. The predictive distribution of Equation (2) is then approximated by sampling multiple times from $q(\mathbf{w})$, and averaging the results.

Despite the approximation, however, VI BNNs still come with several limitations that impede their deployment in practice. First, even in the case of diagonal Gaussian distributions, the number of parameters in the BNN is doubled compared to their deterministic counterpart. Furthermore, approximating the predictive distribution requires multiple sampling procedures and multiple forward passes through the network so that their computational time is orders of magnitude higher than, again, their deterministic counterpart. Finally, despite its greater flexibility, standard Variational Inference struggles to learn very deep BNNs and it is generally limited to more traditional architecture and small-to-medium-size datasets. In the following, we develop a weight-sharing quantisation scheme targeted at BNNs to tackle these limitations.

4 2D GAUSSIAN BAYESIAN NEURAL NETWORK

Consider the vector $\mathbf{w}$ of the BNN's weights, where, at each step of the training process, each weight, w_i $i = 1, \dots, n_\mathbf{w}$, is distributed accordingly to a given Gaussian distribution $\mathcal{N}(\mu_{w_i}, \sigma_{w_i})$. We denote with $\mathcal{N}_{full}$ the full set of weight distributions. Our stochastic weight sharing technique aims at finding a set of 2D Gaussian distributions (which we collectively denote as $\mathcal{N}_{\text{ws}}$) $\mathcal{N}(\mu_1, \Sigma_1), \dots, \mathcal{N}(\mu_k, \Sigma_k)$,[2] with $k \ll n_\mathbf{w}$, and such that $\mathcal{N}_{\text{ws}}$ can be used to approximates the behaviour of $\mathcal{N}_{full}$ in terms of resulting accuracy and uncertainty.

Briefly, we do this by first modelling all the hyperparameters of the $\mathcal{N}_{full}$ distributions through a Gaussian Mixture Model (GMM) (Section 4.2), and then applying alpha-blending to sample weights from the resulting realisations of the GMM (Section 4.3). Additionally, 2DGBNN implements several steps informed by best practice in quantisation for deterministic NNs for further reducing the shared number of weights, including outliers detection (Section 4.1), cluster dimensionality reduction and the merging of similar distributions (Section 4.2). Finally, we will present the overall algorithm for 2DGBNN in Section 4.4.

4.1 Outliers vs. Inliers Classification

The key observation behind the weight-classification stage of our algorithm is that not all the weights of a neural network have an equal impact on the output. Taking inspiration from quantisation techniques for deterministic neural networks (Subedar et al., 2021), we, therefore, do not quantise extreme values in the BNN as those are, likely, particularly influential in the final result. Specifically, we partition the full weight vector $\mathbf{w}$ into two separate vectors, $\mathbf{w}_{in}$ and $\mathbf{w}_{out}$, and only apply weight-sharing to the former. We do this by using two different criteria.

Mean Threshold: Weights associated with a mean with an absolute value greater than a threshold τ (e.g., $\tau = 0.2$) are classified as outliers, i.e.:

$$w_i \in \mathbf{w} \text{ is an outlier if } |\mu_i| > \tau,$$

A discussion of how we chose parameters like τ is provided in Appendix E.

Gradient Threshold: Weights associated with gradient magnitudes exceeding the threshold that places them within the top 1% during backpropagation are also categorized as outliers, as their high gradient values likely signify their

[2] As standard we use σ to denote the one-dimensional standard deviation in the case of 1d Gaussian, and Σ to denote the multidimensional covariance in the case of multidimensional Gaussian.

substantial impact on model performance, i.e.:

$$w_i \in \mathbf{w} \text{ is an outlier if } |\nabla_{w_i}| \text{ is top 1\% of } \{|\nabla_{w_j}|\}_{j=1}^{n_\mathbf{w}}.$$

Algorithm 1 2DGBNN

Input: NN architecture $f^\mathbf{w}$, training data $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$ – $\tau_w, \tau_d, \tau_g, \tau_v$ algorithm thresholds – BNN prior $p(\mathbf{w})$.
Output: Stochastic weight-sharing trained BNN.

Stage 1: Initialise GMM
Initialise μ, σ according to $p(\mathbf{w})$
for each epoch **do** ▷ Pre-training
  Sample weights: $\mathbf{w} = \mu + \sigma \odot \epsilon$, $\epsilon \sim \mathcal{N}(0, \mathbf{I})$
  Update μ, σ by training on $\mathcal{D}$
end for
if $|w_i| > \tau_w$ **or** $|\nabla_{w_i}|$ in top 1% **then** w_i is outlier
  else w_i is inlier ▷ §4.1
end if
Learn GMM on inlier params Θ_{in} (Equation (3))
Stage 2: Refine GMM
for each inlier weight w_i **do**
  Perform §4.3 check on Mahalanobis distance
  if Outside 95th percentile **then**
    w_i is assigned to multiple clusters.
  else w_i is assigned only to the closest Gaussian.
  end if
end for
Apply alpha-blending for ellipse points (Eq. 7)
repeat
  for each pair $(\mathcal{N}_1, \mathcal{N}_2)$ in GMM **do**
    if $W(\mathcal{N}_1, \mathcal{N}_2) < \tau_d$, $\Delta_g < \tau_g$, $\Delta_v < \tau_v$ **then**
      Merge $\mathcal{N}_1$, $\mathcal{N}_2$ using Eqs. (5), (6)
    end if
  end for
until No more Gaussians can be merged
Stage 3: Final BNN Training
for each epoch **do**
  for each weight w_i **do**
    if w_i is inlier **then**
      Sample $w_i \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, \Sigma_k)$
    else
      Use $w_i \sim \mathcal{N}(\mu_{w_i}, \sigma_{w_i}^2)$
    end if
  end for
  Minimising step for $\hat{\mathcal{L}}(\mathcal{D}, q)$ of Eq. (10)
end for

4.2 2DGBNN training

Given the set of distributions of the inlier weights, which we denote as $\mathcal{N}_{in}$, we proceed by clustering their means and variances on the 2-dimensional μ-σ plane. We do this by learning a Gaussian Mixture Model (GMM) of the form: $p((\mu, \sigma)) = \sum_{k=1}^{K} \pi_k \mathcal{N}\left((\mu, \sigma) | \mu_k, \Sigma_k\right)$ over the set of points $\Theta_{in} = \{(\mu_{w_i}, \sigma_{w_i})\}_{i=1}^{n_{\mathbf{w}_{in}}}$. Due to the large volume of points involved in the learning of the GMM (typically, millions of weights), we rely on mini-batch learning for GMMs (Li et al., 2014). This is achieved by sampling random mini-batches $\mathcal{B} \subset \Theta_{in}$, and iteratively minimising the log-likelihood over the mini-batch:

$$\min \sum_{(\mu_i, \sigma_i) \in \mathcal{B}} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}((\mu_i, \sigma_i) \mid \mu_k, \Sigma_k) \right). \quad (3)$$

After the Initial GMM learning, we perform two further reduction steps based on the number of points around each Gaussian, and on the distance between pairs of Gaussians.

Cluster Size Reductions: During the initial Gaussian clustering stage, clusters associated with fewer than 30 weights are identified. Weights within these small clusters are treated as outliers due to their lack of representation within the broader weight distribution. Overall, we empirically find that approximately 1.8% of the total weights of a neural network are generally allocated as outliers.

Merging Gaussians: We merge together Gaussian distributions that are very close to each other. We do this by relying on the distance between Gaussians and their gradients. Specifically, we compute the Wasserstein-2 distance between pairs of distributions as (Jacobs et al., 2023):

$$W_2(\mathcal{N}(\mu_i, \Sigma_i), \mathcal{N}(\mu_j, \Sigma_j))^2 = ||\mu_i - \mu_j||_2^2 + \mathrm{Tr}\left(\Sigma_i + \Sigma_j - 2(\Sigma_i^{1/2}\Sigma_j\Sigma_i^{1/2})^{1/2}\right) \quad (4)$$

If the distance between two Gaussians is less than a given threshold γ, then we inspect the gradient of the network in the weight associated to the cluster centroid and its variance. If those are smaller than two given threshold σ and α then we proceed by merging the two Gaussians into one.[3]

The merger is executed using the following equations (Agueh and Carlier, 2011; Takatsu, 2011):

$$\mu_{\text{merged}} = \frac{\mu_1 + \mu_2}{2} \quad (5)$$

$$\Sigma_{\text{merged}} = \frac{\Sigma_1 + \Sigma_2}{2} + \frac{1}{8}(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T + \frac{1}{2}\left(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2}\right)^{1/2} \quad (6)$$

That ensures that the newly formed Gaussian component accurately reflects the collective distribution characteristics of the initial components while maintaining minimal internal variation.

[3] Details about the thresholds we use in our experiments can be found in the Appendix.

Table 1: Comparison of 2DGBNN and competitive techniques superscripts indicate matching architectures) on **ImageNet1k** dataset. We also provide the number of outliers, ellipses, and Gaussians derived by our method.

Architecture	Method	Accuracy ↓	NLL ↓	ECE ↓	#Outliers	#Ellipses	#Gaussians	#Parameters(M) ↓ / Compression Ratio(%) ↑
ResNet-18	Mutual BNN (Pham et al., 2024)	67.7	1.327	0.1300	-	-	-	23.4M / -
	2GDBNN(ours)	**68.1**	**1.253**	**0.019**	23013	10885	2217	**0.038M / 99%**
ResNet-50	Deep Ensembles (Lakshminarayanan et al., 2017)	77.5	0.877	0.0305	-	-	-	146.7M / -
	Rank-1 BNN(Dusenberry et al., 2020)	77.3	0.886	0.0166	-	-	-	26.0M / -
	ATMC (30 samples) (Heek and Kalchbrenner, 2019)	77.5	0.883	-	-	-	-	768.0M / -
	MCMC (9 samples) BNN (Zhang et al., 2019)	77.1	0.888	-	-	-	-	230.4M / -
	2GDBNN(ours)	75.1	0.961	0.029	37172	56873	3250	**0.101M/ 99%**
ResNet-101	**2DGBNN(ours)**	75.50	0.969	**0.023**	53641	4311	2464	**0.063M** / 99%
VIT-B-16		76.01	0.901	0.064	9765	338329	5440	**0.359M** / 98%

4.3 α-blending (Multi-Clusters) for Weights

Before the final sampling step, we reassess inlier weights $\mathbf{w}_{in}$ by computing their squared Mahalanobis distances to cluster means using, i.e., $D^2 = (w_i - \mu_i)^\top \Sigma_k^{-1}(w_i - \mu_i)$. If a weight's D^2 exceeds 5.991,[4] we reassess its cluster assignment: We do this by relying on α-blending (Mildenhall et al., 2021).

Specifically, for each the $w_i \in \mathbf{w}_{in}$ we compute the subset of GMM's component $\mathcal{N}(\mu_k, \Sigma_k)$, for $k = 1, \ldots, n_i$ such that the above condition on the D^2 is met. We then sample the final value of the weight by the resulting distributions:

$$p(w_i) = \sum_{k=1}^{n_i} \alpha_k \mathcal{N}(\mu_k, \Sigma_k) \tag{7}$$

where α_k is the mixing coefficient, computed as the pdf of (μ_i, σ_i^2) according to $\mathcal{N}(\mu_k, \Sigma_k)$.

4.3.1 Combined Variational Formulation

Finally, we observe that our stochastic weight-sharing technique can be seamlessly integrated within the ELBO formulation for VI training (Nowlan and Hinton, 2018; Zhang et al., 2018). Formally, we assume that $\mathbf{w}_{out}$ and $\mathbf{w}_{in}$ are vectors of pairwise independent weights,[5] which allow us to bound the variational objective as it follows:

$$\mathcal{L}(\mathcal{D}, q) =$$
$$\mathbb{E}_{q(\mathbf{w})}\left[\log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w})\right] - \mathrm{KL}\left(q(\mathbf{w}) \,\|\, p(\mathbf{w})\right) = \tag{8}$$
$$\mathbb{E}_{q(\mathbf{w})}\left[\log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w})\right] - \mathrm{KL}\left(q(\mathbf{w}_{in}) \,\|\, p(\mathbf{w}_{in})\right) -$$
$$\mathrm{KL}\left(q(\mathbf{w}_{out}) \,\|\, p(\mathbf{w}_{out})\right) \approx \mathbb{E}_{q(\mathbf{w})}\left[\log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w})\right] \tag{9}$$
$$- \mathrm{KL}\left(\sum \pi_k \mathcal{N}_k \,\|\, p(\mathbf{w}_{in})\right) - \mathrm{KL}\left(q(\mathbf{w}_{out}) \,\|\, p(\mathbf{w}_{out})\right)$$
$$\geq \mathbb{E}_{q(\mathbf{w})}\left[\log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w})\right] - \sum_k \pi_k \underbrace{\mathrm{KL}\left(\mathcal{N}_k \,\|\, p(\mathbf{w}_{in})\right)}_{\text{GMM KL divergence}} -$$
$$\underbrace{\mathrm{KL}\left(q(\mathbf{w}_{out}) \,\|\, p(\mathbf{w}_{out})\right)}_{\text{Outliers KL divergence}} := \hat{\mathcal{L}}(\mathcal{D}, q), \tag{10}$$

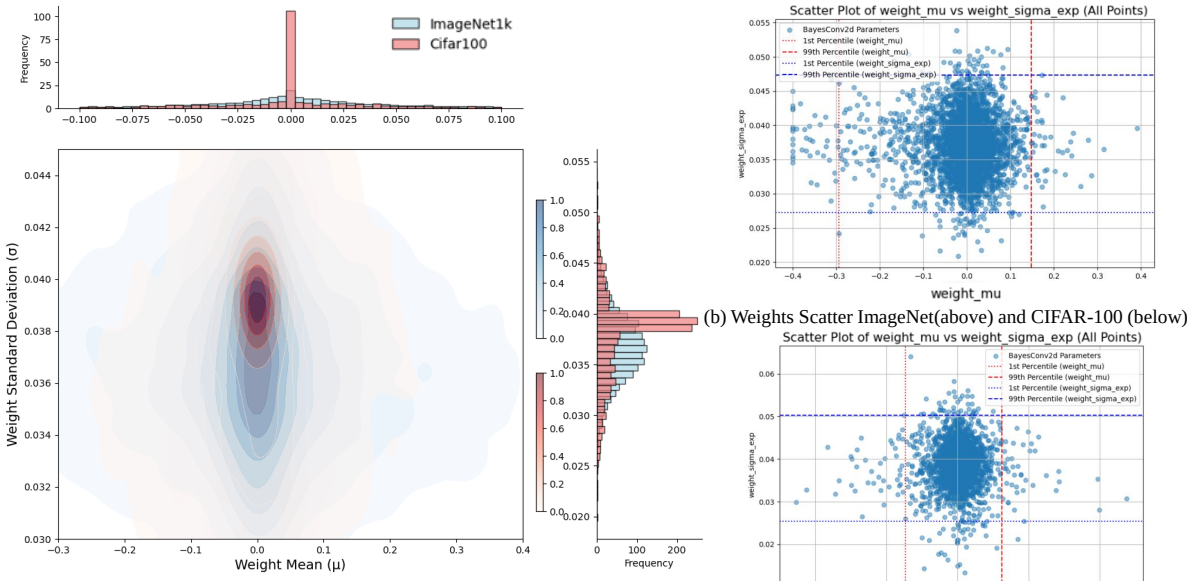
where the equality in Equation (8) is due to the pairwise independence assumption between components of $\mathbf{w}_{in}$ and $\mathbf{w}_{out}$, the approximation of Equation (9) is due to the GMM approximation of the inlier weights, and the final inequality is due to the convexity of the KL divergence. Notice that the resulting value loss function, $\hat{\mathcal{L}}$ is an upper bound on the original loss $\mathcal{L}$ so that its minimisation by means of gradient descent guarantees the improvement of the latter.

4.4 Overall Methodology

The overall methodology is presented in pseudocode form in Algorithm 1. 2DGBNN combines its component parts in three stages. In the first stage, the BNN is initialised with the given prior, a pre-training step is performed and the resulting BNN is used to initialise the GMM clustering. In stage 2, the initial GMM clustering obtained is refined by merging close Gaussians, and by performing α-blending. Finally the BNN is trained by optimising the variational objective on a combination of full weight distributions (for the outliers) and GMM-based weight-sharing (for the inliers).

[4] Corresponding to the 95th percentile of the χ_2^2 distribution which models Mahlanobis distance of multidimensional Gaussians.

[5] This is true for the variational distribution but it is an approximation for the true posterior.



(a) Density and Frequency for ResNet50 in ImageNet and CIFAR-100

(b) Weights Scatter ImageNet(above) and CIFAR-100 (below)

Figure 1: Weight distribution for the BNN prior to stochastic-sharing. Panel (a) shows the density plot for the second convolutional layer of ResNet-50 when trained on ImageNet (in blue) and CIFAR-100 (in red). Panel (b) shows the corresponding scatter plots, including lines for the 1% and 99%.

5 EXPERIMENTS

To validate the effectiveness and scalability of 2DGBNN, we conduct comprehensive experiments using various NN architectures on benchmark image classification datasets. This section details the datasets, models, experimental setup, results, and analysis of our findings. Specifically, we evaluate our method on four common image classification benchmarks: MNIST, CIFAR-10, CIFAR-100, and ImageNet1k; as well as four widely used NN architectures: ResNet-18, ResNet-50 and ResNet-101 and a Vision Transformer (ViT). The hyperparameters used and the details of the training are given in Appendix. Throughout this section, we compare our technique against the results obtained by Deep Ensembles (Lakshminarayanan et al., 2017), Rank-1 BNN (Dusenberry et al., 2020), MCMC BNN (Zhang et al., 2019), ATMC (Heek and Kalchbrenner, 2019), Mutual BNN (Pham et al., 2024), F-SGVB-LRT Nguyen et al. (2024), ABNN (Franchi et al., 2024), LP-BNN (Franchi et al., 2023), IR (Kim et al., 2023), SSVI (Li et al., 2024) and mBCNN (Kong et al., 2023). We evaluate the resulting models in terms of Accuracy, Negative Log-Likelihood (NLL, which measures the model's uncertainty in its predictions), and Expected Calibration Error (ECE, which measures the calibration of predicted probabilities (Guo et al., 2017)). Each experiment is conducted three times with different random seeds, and we report the average results. We conduct experiments with all the aforementioned comparison models, totaling 13 2DGBNN experiments, which include the base models of all comparison methods. Additionally, we perform two quantisation comparison experiments (Section 5.2) and an ablation study (Section 5.3).

5.1 Performance Evaluation

The results obtained with 2DGBNN and those of the state-of-the-art are listed in Tables 1, 2 and 3 for ImageNet1k, CIFAR-100 and CIFAR-10 respectively along with details on the computations performed by 2DGBNN.

In the case of ImageNet1k (Table 1) the comparison is performed against Deep Ensembles (Lakshminarayanan et al., 2017), Rank-1 BNN (Dusenberry et al., 2020), MCMC BNN with 9 samples (Zhang et al., 2019), ATMC (Heek and Kalchbrenner, 2019), and Mutual BNN (Pham et al., 2024). We observe that, in all cases, our method successfully reduces the number of parameters by 3 or 4 orders of magnitudes.While the accuracy is reduced by around 2% in the ResNet-50 case, we do obtain comparable uncertainty estimation as evaluated by NLL and ECE.

Similar results we obtain on the CIFAR-100 dataset (Table 2), comparing against Deep Ensembles (Lakshminarayanan et al., 2017), Rank-1 BNN(Dusenberry et al., 2020), F-SGVB-LRT (Nguyen et al., 2024) and ABNN (Franchi et al., 2024). Our method achieves a substantial reduction in model parameters while maintaining or improving performance compared to other methods at the price of approximately 2% when compared to Deep Ensembles and Rank-1 BNN. For instance, for ResNet-18 we use only 0.019M parameters, which is drastically lower than the 23.4M parame-

Table 2: Comparison of 2DGBNN and competitive techniques (superscripts indicate matching architectures) on **CIFAR-100** dataset. We also provide the number of outliers, ellipses, and Gaussians derived by our method.

Architecture	Method	Accuracy ↑	NLL ↓	ECE ↓	#Outliers	#Ellipses	#Gaussians	#Parameters(M) ↓ / Compression Ratio(%) ↑
ResNet-18	F-SGVB-LRT (Nguyen et al., 2024)	70.1	1.121	0.036	-	-	-	23.4M / -
	SSVI (Li et al., 2024)	75.8	-	0.001	-	-	-	2.32M / 90%
	mBCNN (Kong et al., 2023)	73.7	1.004	0.002	-	-	-	2.86M / 87.8%
	2GDBNN(ours)	74.7	1.053	0.038	14624	260	2387	**0.019M / 99%**
WRN-28-10	Deep Ensembles (Lakshminarayanan et al., 2017)	82.7	0.666	0.021	-	-	-	146M / -
	Rank-1 BNN (Dusenberry et al., 2020)	82.4	0.689	0.012	-	-	-	36.6M / -
	LP-BNN (Franchi et al., 2023)	79.3	-	0.0702	-	-	-	26.8M / 63%
	2GDBNN(ours)	80.5	0.798	0.0432	40354	341	2390	**0.045M / 99%**
ResNet-50	ABNN (Franchi et al., 2024)	74.20	0.828	4.5	-	-	-	54.2M / -
	2GDBNN(ours)	78.1	0.986	0.107	247591	330	1980	**0.251M / 99%**
ResNet-101	**2GDBNN(ours)**	78.4	0.834	0.066	45240	348	3199	**0.052M / 99%**

Table 3: Comparison of 2DGBNN and competitive techniques (superscripts indicate matching architectures) on **CIFAR-10** dataset. We also provide the number of outliers, ellipses, and Gaussians derived by our method.

Architecture	Method	Accuracy ↑	NLL ↓	ECE ↓	#Outliers	#Ellipses	#Gaussians	#Parameters(M) ↓ / Compression Ratio(%) ↑
ResNet-18	F-SGVB-LRT (Nguyen et al., 2024)	90.31	0.262	0.014	-	-	-	23.4M / -
	SSVI (Li et al., 2024)	93.74	-	0.006	-	-	-	1.17M / 95%
	mBCNN (Kong et al., 2023)	93.20	0.220	0.008	-	-	-	0.93M / 96%
	2GDBNN(ours)	91.72	0.305	0.019	123310	67	1569	**0.018M / 99%**
WRN-28-10	Deep Ensembles (Lakshminarayanan et al., 2017)	96.2	0.143	0.020	-	-	-	146M / -
	Rank-1 BNN (Dusenberry et al., 2020)	96.3	0.128	0.008	-	-	-	36.6M / 50.8%
	LP-BNN (Franchi et al., 2023)	95.0	-	0.009	-	-	-	26.8M / 63%
	2GDBNN(ours)	95.2	0.142	0.012	39395	365977	3950	**0.413M / 99%**
ResNet-50	ABNN (Franchi et al., 2024)	95.01	0.160	1.0	-	-	-	54.2M / 25%
	2GDBNN(ours)	93.84	0.223	0.012	129640	**0**	1628	**0.132M / 99%**
ResNet-101	**2GDBNN(ours)**	92.78	0.270	0.015	45240	348	3199	**0.052M / 99%**

ters used by the F-SGVB-LRT model, yet we achieve competitive accuracy. Finally, Table 3 lists analogous results in the context of CIFAR-10, comparing against IR (Kim et al., 2023), F-SGVB-LRT (Nguyen et al., 2024), ABNN (Franchi et al., 2024) and LP-BNN (Franchi et al., 2023).

Additionally to the architectures used for comparisons, the tables report results for ResNet-101 and ViT. In these architectures too, 2DGBNN is able to reduce the number of parameters while obtaining accuracy and uncertainty metrics on par with that of state-of-the-art techniques across the remaining architectures. Interestingly, observing the training results obtained we notice how the distribution of weights in models trained on smaller datasets (CIFAR-100 and CIFAR-10) tends to cluster near zero, as depicted in Figure 1 (b) (down). Conversely, in ImageNet1k the weight distribution is broader, as can be seen from Figure 1 (a)) comparing empirical distributions obtained on CIFAR-100 and ImageNet1k. Notice how this translates to, for example, the ResNet-50 model trained on ImageNet1k dataset to have a significantly greater number of Gaussian and Ellipse weights than when trained on the CIFAR-100 dataset.

5.2 Comparison against Quantisation

We now compare 2DGBNN against quantisation techniques applied to BNNs (Subedar et al., 2021). For this purpose, we remove the pretraining stage of 2DGBNN so to mimic the "vanilla" BNN training employed by Subedar et al. (2021). We use a Gaussian prior with a mean of 0 and a standard deviation of 0.1. Notice that these experiments are limited to CIFAR-10 and MNIST as the vanilla training of BNNs used in Subedar et al. (2021) does not scale to the larger architectures and datasets analysed in the previous section.

The comparative results are presented in Table 4. Accuracy values are very similar across the board, while our technique obtains significantly better uncertainty metrics, except for NLL in the case CIFAR-10.

Table 4: Comparison against the quantisation technique of Subedar et al. (2021) on **CIFAR-10** and **MNIST**.

Datasets	Algorithm	Quantisation technique	Accuracy ↑	NLL ↓	ECE ↓	#Parameters (MB: Megabyte)			
						#Outliers	#Ellipses	#Gaussians	#Parameters
CIFAR-10	BNNs Quantization (Subedar et al., 2021)	ResNet-20 (INT8 SIGMA4)	90.92	0.266	1.778	-	-	-	**0.87 MB**
		ResNet-20 (INT8 SIGMA2)	90.85	0.273	2.547	-	-	-	**0.72 MB**
		ResNet-20 (INT8 SIGMA1)	90.96	0.266	0.711	-	-	-	**0.54 MB**
	2DGBNN	ResNet-20(without pretrained)	90.91	0.303	**0.040**	74181	3634	142	**0.644MB (1.62MB)**
		ResNet-20(with pretrained)	**91.04**	0.303	**0.037**	14624	260	2387	**0.020M / (0.71MB)**
MNIST	BNNs Quantization (Subedar et al., 2021)	ResNet-20 (INT8 SIGMA4)	99.36	0.020	0.215	-	-	-	**0.10 MB**
		ResNet-20 (INT8 SIGMA2)	99.32	0.024	0.277	-	-	-	**0.08 MB**
		ResNet-20 (INT8 SIGMA1)	99.34	0.027	0.351	-	-	-	**0.06 MB**
	2DGBNN	ResNet-20(without pretrained)	**99.52**	0.013	**0.001**	3206	581	237	**0.092MB (0.403MB)**

Table 5: Ablation Study: Impact of Outliers and Ellipses on CIFAR-10 Using ResNet-20

Configuration	#Outliers	#Ellipses	Accuracy (%) ↑	NLL ↓	ECE↓
2DGBNN	✓	✓	90.92	0.265	0.040
Without Ellipse	✓	–	90.43	0.304	0.043
Without Outliers	–	✓	90.18	0.308	0.026
Without Outliers and Ellipse	–	–	90.01	0.318	0.029

In terms of the model size (here compared in Megabytes), the two techniques compare similarly when it comes to the size of trainable parameters (corresponding to the value reported not in brackets for 2DGBNN), with quantisation having a slight edge when only 1 bit is used for encoding the standard deviation. Notice, however, that in small NNs (like the one here analysed) our techniques incur significant storage overhead in that we need to keep an index (encoded in uint8) that assigns each inlier weight to its cluster. When this value is added (size reported in brackets in the Table) quantisation has a significant advantage over our storage requirements. While techniques such as Huffman coding or multi-level index tables can potentially reduce the size of the index vector by several factors, we leave further investigations to future work, and here notice that despite maintaining full precision on the workings of the BNN, weight-sharing quantisation can already obtain comparable results to int8 quantisation. We notice that the two methods are complimentary, and int8 quantisation can further reduce the storage requirements of the outlier weights and GMMs.

5.3 Ablation Study

Table 5 presents an ablation study on CIFAR-10 using ResNet-20 to explore how outliers and ellipses contribute to the performance of 2DGBNN. When both outliers and ellipses are included, the model achieves an accuracy of 90.92%, with the lowest NLL of 0.265 and ECE of 0.040. However, removing either component significantly impacts performance. Without ellipses, the accuracy drops by 0.49% to 90.43%, and excluding outliers reduces it slightly further by 0.25% to 90.18%. When both are removed, the accuracy drops by 0.87% to a low of 90.05%.

6 CONCLUSIONS

We have presented a stochastic weight-sharing quantisation technique based on GMMs specifically tailored to BNNs. In an extensive empirical evaluation, we have seen how our technique can significantly reduce the effective number of parameters of a BNN while obtaining results on par with state-of-the-art in large datasets and architectures such as ImageNet1k and ViT.

Future work will explore how to integrate our method into a fully Bayesian framework and the application of further quantisation for the outlier weights. We have presented a stochastic weight-sharing quantisation technique based on GMMs specifically tailored to BNNs. In an extensive empirical evaluation, we have seen how our technique can significantly reduce the effective number of parameters of a BNN while obtaining results on par with state-of-the-art in large datasets and architectures such as ImageNet1k and ViT. Future work will explore how to integrate our method into a fully Bayesian framework and the application of further quantisation for the outlier weights.

7 ACKNOWLEDGEMENTS

This publication has emanated from research jointly funded by European Union's Horizon Europe 2021–2027 framework programme, Marie Skłodowska-Curie Actions, Grant Agreement No. 101072456 and Taighde Éireann – Research Ireland under grant number 13/RC/2094_2.

References

Achterhold, J., Koehler, J. M., Schmeink, A., and Genewein, T. (2018). Variational network quantization. In *International conference on learning representations*.

Agueh, M. and Carlier, G. (2011). Barycenters in the wasserstein space. *SIAM Journal on Mathematical Analysis*, 43(2):904–924.

Beckers, J., Van Erp, B., Zhao, Z., Kondrashov, K., and De Vries, B. (2023). Principled pruning of bayesian neural networks through variational free energy minimization. *IEEE Open Journal of Signal Processing*.

Bharadiya, J. P. (2023). A review of bayesian machine learning principles, methods, and applications. *International Journal of Innovative Science and Research Technology*, 8(5):2033–2038.

Billah, M. E. and Javed, F. (2022). Bayesian convolutional neural network-based models for diagnosis of blood cancer. *Applied Artificial Intelligence*, 36(1):2011688.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1613–1622.

Bonnet, D., Hirtzlin, T., Majumdar, A., Dalgaty, T., Esmanhotto, E., Meli, V., Castellani, N., Martin, S., Nodin, J.-F., Bourgeois, G., et al. (2023). Bringing uncertainty quantification to the extreme-edge with memristor-based bayesian neural networks. *Nature Communications*, 14(1):7530.

Chien, J.-T. and Chang, S.-T. (2023). Bayesian asymmetric quantized neural networks. *Pattern Recognition*, 139:109463.

Chizat, L., Roussillon, P., Léger, F., Vialard, F.-X., and Peyré, G. (2020). Faster wasserstein distance estimation with the sinkhorn divergence. *Advances in Neural Information Processing Systems*, 33:2257–2269.

De Palma, G., Marvian, M., Trevisan, D., and Lloyd, S. (2021). The quantum wasserstein distance of order 1. *IEEE Transactions on Information Theory*, 67(10):6627–6643.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE.

Doan, B. G., Shamsi, A., Guo, X.-Y., Mohammadi, A., Alinejad-Rokny, H., Sejdinovic, D., Ranasinghe, D. C., and Abbasnejad, E. (2024). Bayesian low-rank learning (bella): A practical approach to bayesian neural networks. *arXiv preprint arXiv:2407.20891*.

Dong, R., Tan, Z., Wu, M., Zhang, L., and Ma, K. (2022). Finding the task-optimal low-bit sub-distribution in deep neural networks. In *International Conference on Machine Learning*, pages 5343–5359. PMLR.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

Dusenberry, M., Jerfel, G., Wen, Y., Ma, Y., Snoek, J., Heller, K., Lakshminarayanan, B., and Tran, D. (2020). Efficient and scalable bayesian neural nets with rank-1 factors. In *International conference on machine learning*, pages 2782–2792. PMLR.

Ferianc, M., Maji, P., Mattina, M., and Rodrigues, M. (2021). On the effects of quantisation on model uncertainty in bayesian neural networks. In *Uncertainty in Artificial Intelligence*, pages 929–938. PMLR.

Forsberg, H., Lindén, J., Hjorth, J., Månefjord, T., and Daneshtalab, M. (2020). Challenges in using neural networks in safety-critical applications. In *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, pages 1–7. IEEE.

Franchi, G., Bursuc, A., Aldea, E., Dubuisson, S., and Bloch, I. (2023). Encoding the latent posterior of bayesian neural networks for uncertainty quantification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Franchi, G., Laurent, O., Leguéry, M., Bursuc, A., Pilzer, A., and Yao, A. (2024). Make me a bnn: A simple strategy for estimating bayesian uncertainty from pre-trained models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12194–12204.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR.

Guo, Y. (2018). A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of*

the IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778.

Heek, J. and Kalchbrenner, N. (2019). Bayesian inference for large scale image classification. *arXiv preprint arXiv:1908.03491*.

Hernández-Lobato, J. M. and Adams, R. (2015). Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International conference on machine learning*, pages 1861–1869. PMLR.

Hinton, G. E. and Neal, R. M. (1995). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

Jacobs, P. M., Patel, L., Bhattacharya, A., and Pati, D. (2023). Memory efficient and minimax distribution estimation under wasserstein distance using bayesian histograms. *arXiv preprint arXiv:2307.10099*.

Kim, K., Ma, E.-Y., Choi, J., and Kim, H. (2023). Inverse-reference priors for fisher regularization of bayesian neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 8264–8272.

Kong, I., Yang, D., Lee, J., Ohn, I., Baek, G., and Kim, Y. (2023). Masked bayesian neural networks: Theoretical guarantee and its posterior inference. In *International Conference on Machine Learning*, pages 17462–17491. PMLR.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30.

Lampinen, J. and Vehtari, A. (2001). Bayesian approach for neural networks—review and case studies. *Neural networks*, 14(3):257–274.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, J., Miao, Z., Qiu, Q., and Zhang, R. (2024). Training bayesian neural networks with sparse subspace variational inference. *arXiv preprint arXiv:2402.11025*.

Li, M., Zhang, T., Chen, Y., and Smola, A. J. (2014). Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670.

Lin, J.-L., Krishnan, R., Ranipa, K. R., Subedar, M., Sanghavi, V., Arunachalam, M., Tickoo, O., Iyer, R., and Kandemir, M. T. (2023). Quantization for bayesian deep learning: Low-precision characterization and robustness. In *2023 IEEE International Symposium on Workload Characterization (IISWC)*, pages 180–192. IEEE.

Louizos, C., Ullrich, K., and Welling, M. (2017). Bayesian compression for deep learning. *Advances in neural information processing systems*, 30.

Marcos, J., Hornero, R., Alvarez, D., Nabney, I. T., Del Campo, F., and Zamarrón, C. (2010). The classification of oximetry signals using bayesian neural networks to assist in the detection of obstructive sleep apnoea syndrome. *Physiological measurement*, 31(3):375.

Michelmore, R., Wicker, M., Laurenti, L., Cardelli, L., Gal, Y., and Kwiatkowska, M. (2020). Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 7344–7350. IEEE.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2021). Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106.

Minka, T. P. (2001). Expectation propagation for approximate bayesian inference. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 362–369.

Nguyen, V.-A., Vuong, T.-L., Phan, H., Do, T.-T., Phung, D., and Le, T. (2024). Flat seeking bayesian neural networks. *Advances in Neural Information Processing Systems*, 36.

Nowlan, S. J. and Hinton, G. E. (2018). Simplifying neural networks by soft weight sharing. In *The mathematics of generalization*, pages 373–394. CRC Press.

Park, N., Lee, T., and Kim, S. (2021). Vector quantized bayesian neural network inference for data streams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9322–9330.

Perrin, M., Guicquero, W., Paille, B., and Sicard, G. (2024). Hardware-aware bayesian neural architecture search of quantized cnns. *IEEE Embedded Systems Letters*.

Pham, V. C., Nguyen, C. C., Le, T., Phung, D., Carneiro, G., and Do, T.-T. (2024). Model and feature diversity for bayesian neural networks in mutual learning. *Advances in Neural Information Processing Systems*, 36.

Roth, W. and Pernkopf, F. (2018). Bayesian neural networks with weight sharing using dirichlet processes. *IEEE transactions on pattern analysis and machine intelligence*, 42(1):246–252.

Sharma, H. and Jennings, E. (2021). Bayesian neural networks at scale: a performance analysis and pruning study. *The Journal of Supercomputing*, 77(4):3811–3839.

Soudry, D., Hubara, I., and Meir, R. (2014). Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. *Advances in neural information processing systems*, 27.

Subedar, M., Krishnan, R., Kashyap, S. N., and Tickoo, O. (2021). Quantization of bayesian neural networks and its effect on quality of uncertainty. In *Workshop on Uncertainty and Robustness in Deep Learning, ICML*.

Subia-Waud, C. and Dasmahapatra, S. (2024). Probabilistic weight fixing: Large-scale training of neural network weight uncertainties for quantisation. *Advances in Neural Information Processing Systems*, 36.

Swiatkowski, J., Roth, K., Veeling, B., Tran, L., Dillon, J., Snoek, J., Mandt, S., Salimans, T., Jenatton, R., and Nowozin, S. (2020). The k-tied normal distribution: A compact parameterization of gaussian mean field posteriors in bayesian neural networks. In *International conference on machine learning*, pages 9289–9299. PMLR.

Takatsu, A. (2011). Wasserstein geometry of gaussian measures.

Treiss, A., Walk, J., and Kühl, N. (2021). An uncertainty-based human-in-the-loop system for industrial tool wear analysis. In *Machine Learning and Knowledge Discovery in Databases. Applied Data Science and Demo Track: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part V*, pages 85–100. Springer.

Ullrich, K., Meeds, E., and Welling, M. (2017). Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*.

Van Baalen, M., Louizos, C., Nagel, M., Amjad, R. A., Wang, Y., Blankevoort, T., and Welling, M. (2020). Bayesian bits: Unifying quantization and pruning. *Advances in neural information processing systems*, 33:5741–5752.

Vehtari, A. and Lampinen, J. (1999). Bayesian neural networks for industrial applications. In *SMCia/99 Proceedings of the 1999 IEEE Midnight-Sun Workshop on Soft Computing Methods in Industrial Applications (Cat. No. 99EX269)*, pages 63–68. IEEE.

Welling, M. and Teh, Y. W. (2011). Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 681–688.

Wicker, M., Laurenti, L., Patane, A., Paoletti, N., Abate, A., and Kwiatkowska, M. (2024). Probabilistic reach-avoid for bayesian neural networks. *Artificial Intelligence*, page 104132.

Yang, Y., Bamler, R., and Mandt, S. (2020a). Variational bayesian quantization. In *International Conference on Machine Learning*, pages 10670–10680. PMLR.

Yang, Z., Wang, Y., Han, K., Xu, C., Xu, C., Tao, D., and Xu, C. (2020b). Searching for low-bit weights in quantized neural networks. *Advances in neural information processing systems*, 33:4091–4102.

Zhang, C., Bütepage, J., Kjellström, H., and Mandt, S. (2018). Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):2008–2026.

Zhang, R., Li, C., Zhang, J., Chen, C., and Wilson, A. G. (2019). Cyclical stochastic gradient mcmc for bayesian deep learning. *arXiv preprint arXiv:1902.03932*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. Yes
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. Yes

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. Yes
   (b) Complete proofs of all theoretical results. Not Applicable
   (c) Clear explanations of any assumptions. Yes

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). Yes
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Yes
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). Yes
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). Yes

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. Yes
   (b) The license information of the assets, if applicable. Yes
   (c) New assets either in the supplemental material or as a URL, if applicable. Not Applicable
   (d) Information about consent from data providers/curators. Not Applicable
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content.Not Applicable

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. Not Applicable
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not Applicable
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not Applicable

Stochastic Weight Sharing for Bayesian Neural Networks Supplementary Materials

A Posterior Visualisation

The posterior distribution of weights provides direct insight into the behaviour of 2D Gaussian Bayesian Neural Network (2DGBNN). To present this, we visualise the posterior distribution of ResNet-18 trained on the CIFAR-10 dataset in Figure 2 (left plot). In the figure, red points represent outlier weights that retain their individual values instead of participating in weight sharing, while yellow points denote ellipse weights shared through alpha-blending. Blue points indicate the centres of the Gaussian distributions after training (note that for simplicity of visualisation, we show only 20 outlier weights and 50 ellipse weights).

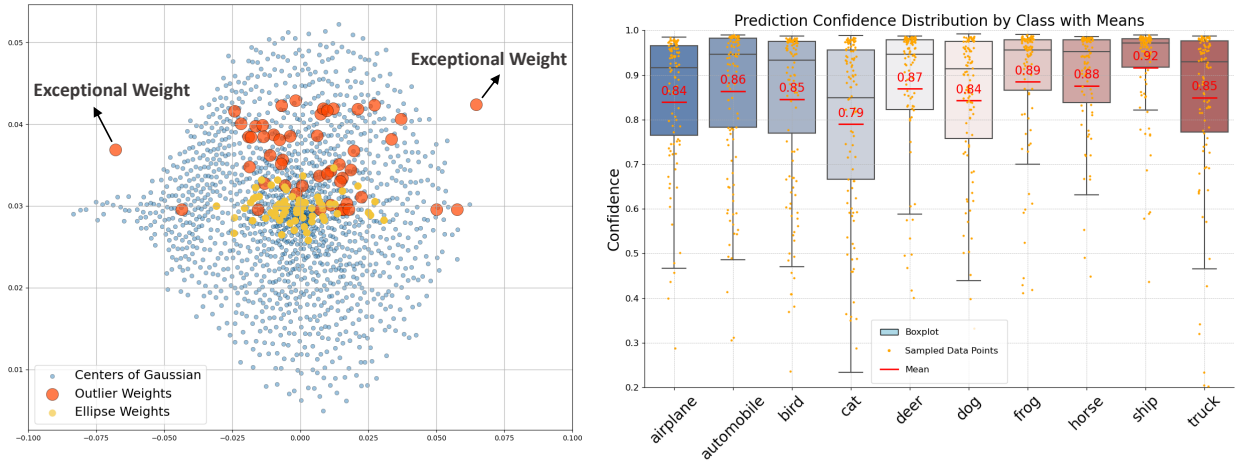


Figure 2: Left: Posterior distribution of weights in ResNet-18 with CIFAR-10 using 2D Gaussian Bayesian Neural Network (2DGBNN), depicting Gaussian centers (blue), Ellipse weights (yellow), and Outlier weights (red). Exceptional weights (outliers). Right: Confidence distribution of predictions on the CIFAR-10 dataset. Mean confidence values are shown above each boxplot.

Additionally, the covariance resulting from the stochastic weight sharing is visualised in Figure 3. To clarify the behaviour of the model's uncertainty, we visualise the outputs of ResNet-18 on the CIFAR-10 dataset in Figure 2 (Right). This figure provides a view of the confidence distributions across all ten CIFAR-10 classes.

B Stochastic Weight Initialisation

Model initialisation always affects training and convergence. Here, we demonstrate the initial state of 2D Gaussian, including the visualisation of the covariance and center, as well as weights in different colours. Figure 3 visualises the 2D Gaussian of the second convolutional layer of ResNet-18 in CIFAR-10 dataset in the μ-σ space.

The ellipse weights, (the ones outside two standard deviations from any one cluster) are of crucial importance for inference, as most of them represent error points introduced by the Gaussian Mixture Model (GMM). To address this, we reassess their properties by reallocating them to multiple Gaussian distributions using alpha blending.

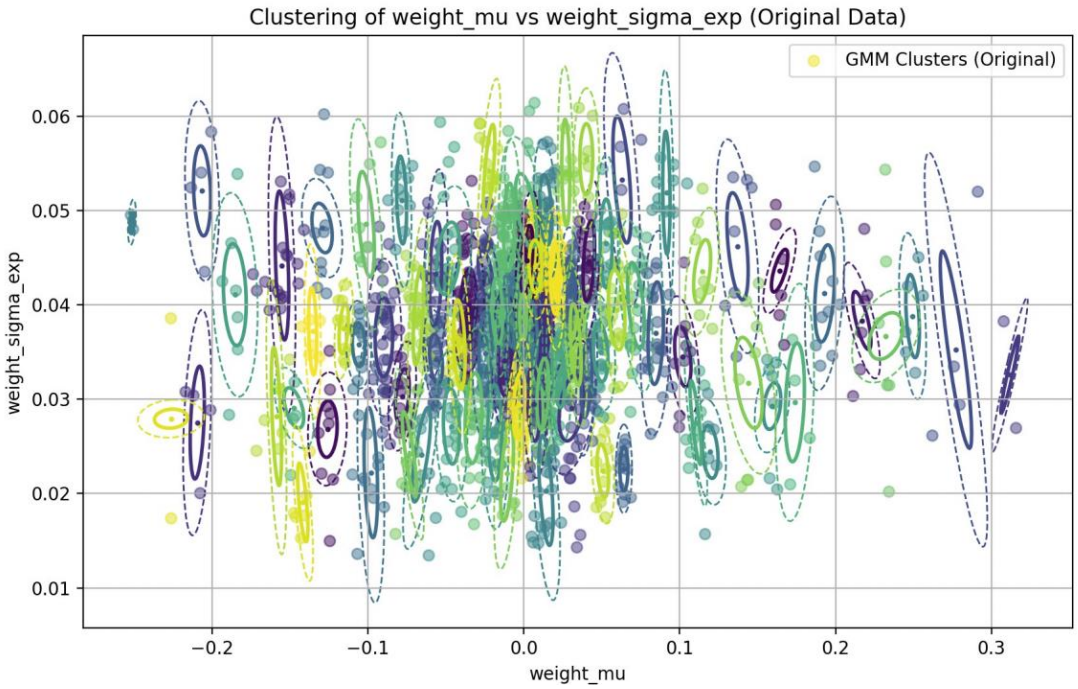


Figure 3: Initial 2D Gaussian of the second convolutional layer of ResNet-18 in CIFAR-10 dataset. Different colors represent the different clusters. The solid line is the first variance area, and the dotted line is the second variance area.

Table 6: Training on NVIDIA Tesla V100 and Inference on NVIDIA RTX 4070 Laptop: A Comparative Analysis of Our 2DGBNNs Method and "Bayes by Backprop"

Method	**Training Time**	**Inference Time (CIFAR-10)**
Our Method (2DGBNNs)	Pre-training: 60–70 min	1024 images: 12.75 sec
	Quantisation: 1.5–2 min	Total: 151.86 sec
	Re-training: 30–35 min	
	Using existing pre-trained model: ∼half time	
Bayes by Backprop	Total: 5–5.5 hours	1024 images: 73.62 sec
(ResNet-18, CIFAR-10)	(140–150 epochs, 10 samples, batch size 256)	Total: 882.41 sec

C Training and Inference Time Comparison

The overall training time will depend on the architecture and dataset employed, as well as the hardware used for training. Here, we give figures on the case of 2DGBNNs with the architecture of ResNet-18 trained for CIFAR-10, on a Tesla V100. The inference was executed on an RTX 4070 (Laptop) with a batch size of 1024.

We notice, that training is a one-off cost and that the main advantages will be in the computational time at inference time. From the Table.6, we compared training and inference times between Our Method (2DGBNNs) and Bayes by Backprop for ResNet-18 on CIFAR-10. Our method significantly reduces training time (50–70 minutes pre-training, 1.5–2 minutes quantization, 30–35 minutes re-training) compared to 5–5.5 hours for Bayes by Backprop. Additionally, inference time is much faster with 2DGBNNs (151.86 seconds) versus Bayes by Backprop (882.41 seconds), demonstrating the efficiency of the multi-stage training approach.

D Out of Distribution Detection Task

Table 7: OOD Detection Results: ResNet-18-2GDBNN trained on CIFAR-10, evaluated on the SVHN dataset.

Metric	Value
AUROC (Area Under ROC Curve)	0.8867
AUPR (Area Under Precision-Recall Curve)	0.8397
FPR85 (False Positive Rate at 85% True Positive Rate)	0.2932

In this experiment, we evaluated the performance of our model on the Out-Of-Distribution (OOD) detection task. The model, ResNet-18-2GDBNN, was trained on the CIFAR-10 dataset, with a compression ratio of 99% and a parameter count of 0.018M. For OOD detection, we used the SVHN (Street View House Numbers) dataset. The result is shown in the Table 7. It achieved an AUROC (Area Under the ROC Curve) of 88.67%. Additionally, the AUPR (Area Under the Precision-Recall Curve) reached 83.97%, And FPR85 (False Positive Rate at 85% True Positive Rate) got 29.32%.

E Hyper-parameters Discussion

The 2GDBNN involves several hyper-parameters. Among these, τ_w, as shown in Algorithm I, line 6, is the most critical and sensitive parameter, playing a key role in the selection of outlier and inlier weights. To demonstrate the process of selecting τ_w, we conducted an empirical experiment, the results of which are detailed in Table 8.

Table 8: Experimental results for varying the value of τ_w on 2DGBNNs with ResNet-20 architecture on CIFAR-10.

τ_w	Accuracy	NLL	ECE	Outliers	Ellipses	Gaussians	Parameters
0.1	91.14%	0.376	0.0197	223046	4501	2804	0.238M
0.2	91.04%	0.303	0.037	14624	260	2387	0.020M
0.4	90.53%	0.396	0.0279	101970	4369	120	0.111M
0.6	89.78%	0.322	0.038	12211	3795	161	0.020M

We empirically selected the hyper-parameters for our methodology through a combination of grid search and trial-and-error over the training set. We performed an experimental evaluation of the 2DGBNNs with the ResNet-20 architecture on the CIFAR-10 dataset to study the effect of varying the value of $\tau_w = 0.2$.

These results demonstrate that as τ_w increases, fewer weights are classified as outliers, which reduces model complexity. However, this also slightly decreases accuracy and increases errors, with $\tau_w = 0.2$ achieving the best overall trade-off.

F Results with Deterministic Neural Networks

In this section, we present the results of the experiments discussed in the main paper but obtained using deterministic neural networks, across three datasets and five architectures. These details are shown in the Table 9. These deterministic networks are used as prior for the centring of GMMs in the main comparisons discussed in the main paper (Section 5.1).

Table 9 displays the performance metrics for deterministic models used as initialization priors for the stochastic counterparts in our 2DGBNNs framework. For each of the three datasets—ImageNet-1k, CIFAR-100, and CIFAR-10—we report the accuracy, Negative Log Likelihood (NLL), Expected Calibration Error (ECE), and the number of parameters. The models evaluated include variations of ResNet (ResNet-18, ResNet-50, ResNet-101) and other architectures such as VIT-B-16 and WRN-28-10.

Table 9: Deterministic neural networks used as prior for the Initialisation 2GDBNN in the experiments discussed in Section 5.1.

Method	Dataset	**Accuracy ↑**	**NLL ↓**	**ECE ↓**	**#Parameters (M: million)**
ImageNet-1k	ResNet-18	69.70	1.265	0.027	11.7M
	ResNet-50	76.10	0.989	0.035	25.6M
	ResNet-101	77.30	0.936	0.936	44.5M
	VIT-B-16	81.07	0.856	0.056	86.0M
CIFAR-100	ResNet-18	77.10	1.038	0.114	11.7M
	ResNet-50	79.20	0.950	0.054	25.6M
	ResNet-101	80.02	0.849	0.095	44.5M
	WRN-28-10	81.41	0.766	0.045	53.6M
CIFAR-10	ResNet-20	91.84	0.246	0.031	0.27M
	ResNet-18	93.21	0.201	0.022	11.7M
	ResNet-50	94.81	0.211	0.010	25.6M
	ResNet-101	94.70	0.849	0.095	44.5M
	WRN-28-10	95.92	0.131	0.010	53.6M

The table is structured to highlight the performance across multiple architectures and datasets, facilitating a direct comparison. Higher accuracies and lower NLL and ECE values indicate better model performance. For instance, VIT-B-16 on ImageNet-1k achieves an accuracy of 81.07% with the lowest ECE of 0.056 among its dataset counterparts. Similarly, WRN-28-10 shows superior performance on CIFAR-10 with the highest accuracy of 95.92% and a remarkably low ECE of 0.010. The number of parameters, reported in millions, also provides insight into the model complexity, ranging from 0.27M for ResNet-20 on CIFAR-10 to 86.0M for VIT-B-16 on ImageNet-1k.

The 2DGBNN models on ImageNet1k showed a decrease in accuracy, with our ResNet-50 configuration dropping from the benchmark high of 77.5% to 75.10%. This reduction was coupled with a substantial decrease in the number of parameters—from models requiring up to 25.6 million parameters to just 0.101 million. Despite these changes, the increases in NLL and ECE were minimal and within acceptable ranges, indicating that the models maintain satisfactory predictive performance and calibration despite the reduced complexity. Similar trends were observed in the CIFAR-100 dataset, where our WRN-28-10 model's accuracy decreased from 81.41% to 80.5%, while significantly reducing the parameter count to only 0.045 million from 53.6 million. The NLL and ECE metrics, although slightly elevated, remained competitive, affirming the effective uncertainty estimation capabilities of the models despite their reduced complexity.

G Additional Background

KL Divergence between Gaussians The KL divergence between two Gaussian distributions $q(w) = \mathcal{N}(w|\mu_q, \sigma_q^2)$ and $p(w) = \mathcal{N}(w|\mu_p, \sigma_p^2)$ is given by:

$$\mathrm{KL}(q(w)\|p(w)) = \frac{1}{2}\left(\frac{\sigma_q^2}{\sigma_p^2} + \frac{(\mu_p - \mu_q)^2}{\sigma_p^2} - 1 + \ln\frac{\sigma_p^2}{\sigma_q^2}\right) \tag{11}$$

This formula can be used to compute the KL divergence terms in the ELBO expression for both inliers and outliers.

Expected Log-Likelihood Computation

The expected log-likelihood term involves an expectation over the variational posterior:

$$\mathbb{E}_{q(\mathbf{w})}\left[\log p(\mathbf{y}|\mathbf{X},\mathbf{w})\right] = \int q(\mathbf{w}) \log p(\mathbf{y}|\mathbf{X},\mathbf{w})\, d\mathbf{w} \tag{12}$$

In practice, this integral is intractable and is approximated using Monte Carlo sampling.

H Wasserstein-based 2D Gaussian Merging

In the context of comparing two Gaussian distributions, the Wasserstein distance provides a meaningful way to measure the distance between probability distributions. Specifically, for two 2D Gaussian distributions $\mathcal{N}(\mu_1, \Sigma_1)$ and $\mathcal{N}(\mu_2, \Sigma_2)$, where μ_1, μ_2 are the means and Σ_1, Σ_2 are the covariance matrices, the Wasserstein-2 distance is given by the following formula:

$$W_2(\mathcal{N}(\mu_1, \Sigma_1), \mathcal{N}(\mu_2, \Sigma_2))^2 = ||\mu_1 - \mu_2||_2^2 + \mathrm{Tr}(\Sigma_1 + \Sigma_2 - 2(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}) \tag{13}$$

Where $||\mu_1 - \mu_2||_2$ is the Euclidean distance between the means of the two distributions, Tr is the trace operator, which sums the diagonal elements of a matrix, $\Sigma_1^{1/2}$ refers to the matrix square root of the covariance matrix Σ_1.

The Wasserstein distance threshold is set at 1.5×10^{-7}, with further discussion on the I.1. If the distance between Gaussian components falls below this, gradient information is further considered, ensuring a precise analysis of component similarity.

This method ensures that the newly formed Gaussian component accurately reflects the collective distribution characteristics of the initial components while maintaining minimal internal variation.

I Sub-module Discussion and Hyper-parameter Configuration

I.1 Effectiveness of 2D Gaussian Merging Discussion

In this section, we discuss the effectiveness of the merging of 2D Gaussian distributions in our method. Experimental results analysing its effect are listed in Table 10. For the ResNet20 model on the CIFAR-10 dataset, after merging the 2D Gaussian, the accuracy improved from 88.77% to 91.02%, the NLL decreased from 0.3792 to 0.3066, and the ECE reduced from 0.0500 to 0.0374. For the ResNet18 model on the CIFAR-100 dataset, although the improvement is smaller, the accuracy still increased from 74.50% to 74.63%, and the NLL decreased from 1.0555 to 1.0535. These results demonstrate the effectiveness of our 2D Gaussian merging method in enhancing model performance and calibration.

Table 10: Performance of ResNet20 on CIFAR-10 and ResNet18 on CIFAR-100 before and after merging Gaussian distributions.

Model	Dataset	Method	Accuracy (%)	NLL	ECE
ResNet20	CIFAR-10	Before Merging Gaussian	88.77	0.3792	0.0500
		After Merging Gaussian	91.02	0.3066	0.0374
ResNet18	CIFAR-100	Before Merging Gaussian	74.50	1.0555	0.0391
		After Merging Gaussian	74.63	1.0535	0.0400

I.2 Data Augmentation

To improve the robustness and generalization of our models, we applied a series of data augmentation techniques during training on the ImageNet-1K, CIFAR-100, and CIFAR-10 datasets. Table 11 summarises the specific augmentation methods used for each dataset. For the ImageNet-1K dataset, we applied random resized cropping to obtain images of size 224×224 pixels. This was followed by random horizontal flipping with a probability of 50% to augment the dataset with mirrored images. Color jittering was used to adjust the brightness, contrast, saturation, and hue of the images with factors of 0.4, 0.4,

Table 11: Summary of data augmentation techniques applied to each dataset.

Dataset	Data Augmentation Techniques
ImageNet-1K	Random resized crop to 224×224 pixels, Random horizontal flip, Color jittering (brightness=0.4, contrast=0.4, saturation=0.4, hue=0.1), Conversion to tensor, Normalization
CIFAR-100	Conversion to tensor, Padding of 4 pixels (reflection mode), Random crop to 32×32 pixels, Random horizontal flip, Conversion to tensor, Normalization
CIFAR-10	Random crop to 32×32 pixels, Random horizontal flip, Conversion to tensor, Normalization

0.4, and 0.1, respectively. The images were then converted to tensors and normalised using the standard mean and standard deviation values for ImageNet. In the case of the CIFAR-100 dataset, we started by converting the images to tensors. We then padded the images with 4 pixels on each side using reflection mode to preserve edge information. After padding, we performed a random crop to 32×32 pixels, followed by random horizontal flipping to introduce mirror variations. The images were converted back to tensors and normaliseded accordingly.

For the CIFAR-10 dataset, the augmentation process involved a random crop to 32×32 pixels, which helps in teaching the model to be invariant to translations. We also applied random horizontal flipping to include mirrored versions of the images. Finally, the images were converted to tensors and normalised to standardise the input data.

I.3 Hyperparameters for training the deterministic network

In our experiments, similar to those conducted by previous researchers, we standardised the hyperparameters across all models in the initial stage. This approach was applied to various models including ResNet-18, ResNet-50, ResNet-101, and VIT. The hyperparameters used are as follows:

Hyperparameter	Variable	Default Value
Batch Size	`-b`	256
Warm-up Phases	`-warm`	2
Learning Rate	`-lr`	0.1
Resume Training	`-resume`	`False`
Total Epochs	`-EPOCH`	250
Milestones	`-MILESTONES`	[30, 60, 90, 120, 150, 200]
Weight Decay	`-MultiStepLR`	5e-4
Training Mean	`-TRAIN_MEAN`	(0.5071, 0.4865, 0.4409)
Training Std	`-TRAIN_STD`	(0.2673, 0.2564, 0.2761)

Table 12: Summary of hyperparameters in the neural network training configuration

The table summarizes the standardised hyperparameters used across all models in our neural network training configurations, mirroring settings from previous research. It outlines common parameters such as batch size, warm-up phases,

learning rate, and total epochs, alongside specific settings like weight decay and learning rate milestones.

I.4 Discussion the Initialisation of σ

We experimented with several different initialisation methods for our models, including initialisation via a specific function as detailed by Lee et al., random generation, and Gaussian distribution, among others. According to our experimental results, we ultimately adopted the following initialisation methods for our neural network parameters.

For the weight parameter weight_sigma, we used the Xavier uniform initialisation with a gain of 0.01, defined as:

$$\mathbf{W} \sim \mathcal{U}\left(-\frac{g}{\sqrt{n_{\text{in}} + n_{\text{out}}}}, \frac{g}{\sqrt{n_{\text{in}} + n_{\text{out}}}}\right) \tag{14}$$

where $g = 0.01$, n_{in} is the number of input units, n_{out} is the number of output units, and $\mathcal{U}(a, b)$ denotes a uniform distribution between a and b.

For the bias parameter bias_sigma, we initialised it using a normal distribution with a mean of 0.0 and a standard deviation of 0.001:

$$\mathbf{b} \sim \mathcal{N}(0, 0.001^2) \tag{15}$$

In our neural network, we assign different learning rates to different parameters. Table 13 summarises these hyperparameters.

Table 13: Summary of Hyperparameters Used in Our Experiments

Hyperparameter	**Symbol**	**Value**
Learning rate for weight and bias μ	$\eta_{\text{weight\_mu}}$	1×10^{-4}
Learning rate for weight and bias σ	$\eta_{\text{weight\_sigma}}$	1×10^{-2}

We set the learning rates for the parameters as follows: the learning rates for **weight_mu** and **bias_mu** are both $\eta_{\text{weight\_mu}} = 1 \times 10^{-4}$; the learning rates for **weight_sigma** and **bias_sigma** are both $\eta_{\text{weight\_sigma}} = 1 \times 10^{-3}$.

I.5 Hyper-parameters for training Bayesian Neural Network

In our experiments, we utilise several hyperparameters that are crucial for the performance and convergence of our Bayesian neural network (BNN) model. These hyperparameters are carefully selected based on empirical studies to balance computational efficiency and model accuracy. Table 14 summarises the hyperparameters used in our experiments.

In the KMeans clustering algorithm, we initially use $K = 2000$ clusters. Outliers in the weight values are identified using a threshold $T_w = \pm 0.2$. Any weight value exceeding this threshold is considered an outlier. This threshold is chosen based on the empirical distribution of the weights after initial training. Similarly, outliers in the gradients are identified by selecting the top $P_g = 1\%$ of gradient magnitudes.

A minimum cluster size of $N_{\text{min}} = 30$ is enforced to ensure statistical significance in the clustering results. Clusters with fewer than N_{min} samples are considered invalid and their associated weights are treated as outliers. This prevents the model from being influenced by clusters that may represent noise or insignificant patterns.

During the BNN training, we use a learning rate of $\eta_{\text{BNN}} = 1 \times 10^{-5}$, which is lower than the initial learning rate used in the preliminary training. The smaller learning rate is necessary to accommodate the Bayesian updates and to ensure that the posterior distributions over the weights converge properly.

In the predictive function, we draw $N_s = 30$ samples from the posterior distribution to estimate the predictive mean and uncertainty. The Expected Calibration Error (ECE) is computed using $N_b = 15$ bins. The Mahalanobis distance threshold $T_M = 5.991$ corresponds to the chi-squared distribution value with 2 degrees of freedom at the 95% confidence level. The Mahalanobis distance for a data point x with respect to a Gaussian distribution with mean μ and covariance Σ is calculated as:

$$D_M^2 = (x - \mu)^\top \Sigma^{-1} (x - \mu). \tag{16}$$

Table 14: Summary of Hyperparameters Used in Our Experiments

Hyperparameter	**Symbol**	**Value**
Initial learning rate for μ	η_μ	1×10^{-4}
Number of epochs	E	200
Number of clusters in KMeans	K	6000
Outlier threshold (weight value)	T_w	± 0.2
Outlier threshold (gradient percentile)	P_g	Top 1%
Minimum samples per cluster	N_{min}	20
Learning rate in BNN training	η_{BNN}	1×10^{-5}
Number of samples in predictive function	N_s	30
Number of bins in ECE computation	N_b	15
Wasserstein distance threshold	T_W	1×10^{-2}
Mahalanobis distance threshold	T_M	5.991
Number of nearest Gaussians	k	5

Points with a Mahalanobis distance greater than T_M are considered outliers.

In handling outliers, we consider the $k = 5$ nearest Gaussian components for each outlier point. This allows us to reassign outlier weights to the most probable Gaussian components based on their proximity in the parameter space.