Evolutionary Intelligence (2024) 17:2515–2536
https://doi.org/10.1007/s12065-023-00900-9

RESEARCH PAPER

Gaussian mixture models for training Bayesian convolutional neural networks

Bakhouya Mostafa[1] · **Ramchoun Hassan**[1,2] · **Hadda Mohammed**[1] · **Masrour Tawfik**[1,3]

Received: 26 March 2023 / Revised: 6 December 2023 / Accepted: 17 December 2023 / Published online: 25 January 2024

Abstract
Bayes by Backprop is a variational inference method based on the reparametrization trick to assure backpropagation in Bayesian neural networks. Generally, the approximate distributions used in Bayes by backprop method are made unimodal to facilitate the use of the reparametrization trick. But frequently, the modelling of some tasks requires more sophisticated distributions. This paper describes the Bayes by Backprop algorithm with a multi-model distribution for training Bayesian convolutional neural networks. Specifically, we illustrate how to reparameterize the CNN parameters for a Gaussian mixture model. We then show that the results compare favourably to existing variational algorithms on various classification datasets. Finally, we illustrate how to use this distribution to estimate epistemic and aleatoric uncertainty.

Keywords Convolution neural network · Variational inference · Bayesian neural network · Parameter estimation · Gaussian mixture model · Uncertainty

1 Introduction

Deep learning has emerged as an extension and development of machine learning for providing appropriate solutions to challenging tasks. Deep learning algorithms have provided innovative solutions in many fields, including computer vision, industrial and financial engineering, biomedical engineering, healthcare, security, etc [22, 23, 28, 52, 56, 63–65]. Nowadays, Convolutional Neural Network (CNN) is among the most common deep learning models, especially in pattern recognition and image classification fields due to its compatibility with image architecture. CNN has shown outstanding results, especially in large image classification datasets (Image Net) [2, 43, 51, 59].

✉ Ramchoun Hassan
h.ramchoun@umi.ac.ma

Bakhouya Mostafa
mos.bakhouya@edu.umi.ac.ma

Hadda Mohammed
mohamedhdda@gmail.com

Masrour Tawfik
t.masrour@ensam.umi.ac.ma

1 Department of Mathematics and Computer Science, ENSAM-Meknes, Moulay Ismail University of Meknes, 50050 Meknes, Morocco

2 National School of Business and Management, Moulay Ismail University of Meknes, 50050 Meknes, Morocco

3 Mathematics, Computer Science and Engineering Department, University of Quebec at Rimouski, Rimouski, Canada

1.1 Problem statement

CNN remains limited because it can only provide point estimates of parameters and outcomes. This constraint leads to decisions being taken with a high degree of certainty, which can lead to undesirable results, particularly in situations where decisions require a high degree of caution and credibility. The drawback arises from the fact that a CNN is unable to capture model uncertainty; it consistently produces a result but fails to indicate the reliability or correctness of that result [34].

1.2 Current status

The Bayesian approach in deep learning algorithms can be used as an alternative to the deterministic approach to address this problem. [37]. The Bayesian approach provides a probabilistic interpretation for deep learning models that allows the determination of model uncertainty, which increases the credibility of neural network outcomes [1, 14, 32, 45].

A Bayesian Convolutional Neural Network (BCNN) is a form of artificial neural network in which the CNN parameters are represented as stochastic components of the network.

The BCNN is characterized not only by its capacity to quantify uncertainty in the network but also by its ability to discriminate between its two forms, which are epistemic and aleatoric uncertainty [14, 71]. Nevertheless, applying the Bayesian method to CNN remains challenging since it requires the computation of an integral over all possible values of CNN parameters to determine the posterior distribution of these models. To overcome this problem, we can infer this distribution using statistical estimates known as "approximate inferences" [19, 27, 27, 48, 60, 62]. In this context, there are many inferential estimation techniques used in deep learning algorithms, the most notable of which are Monte Carlo techniques and Variational Inference.

Monte Carlo methods are stochastic estimations that approximate expectations numerically by randomly generating samples from probability distributions.

Variational inference is an optimization technique that provides analytical estimates of the posterior distribution using another simple distribution [7, 60, 69].

1.3 Research hypothesis

In practical applications [3, 41, 46], posterior distributions are often complex and require more flexible models for accurate estimation. Therefore, this paper aims to approximate the posterior distribution using the Bayes by Backprop method by employing a multimodal distribution, specifically a mixture of factorized Gaussian distributions, as the variational distribution. This involves sampling the convolutional parameters from a mixture of K fully factorized Gaussian distributions at each iteration, where K represents the number of mixture components.

Multimodal distributions perform better than unimodal distributions in capturing complex patterns and variations in the posterior distribution.

1.4 The main contributions

The contributions of this paper are summarized as follows:

- We employ the Bayes-Backprop method to train convolutional neural networks (CNNs) by employing a mixture of factorized Gaussian distributions as the variational distribution.
- We show that the reparametrization of a GMM involves reparametrizing each Gaussian component independently. This process includes introducing a standard normal random variable and using it to reparameterize each component of the GMM.
- We apply our proposed method to train CNNs on various datasets. Subsequently, we conduct a comparative analysis between our approach and previous methods to evaluate the obtained results.
- We study the estimation of both aleatoric and epistemic uncertainties using a Gaussian mixture model (GMM) as the variational distribution. Through the experimental results, we demonstrate that as training accuracy increases, uncertainty decreases, resulting in more reliable decision-making by the network.

The remaining sections of this paper were ordered as follows: the second section provided an overview of most of the research related to this work, while Sect. 3 provided the background and tools that we will need in this research, including a description of the BCNN and the most important approaches used in it. Section 4 describes our contribution, which is represented by the application of the Bayes by backprop technique to train CNNs using mixture models. Then, in Sect. 5, we show the experimental results of this method and attempt to compare it to previous works. Finally, Sect. 6 summarizes this paper.

2 Related works

As previously stated, there are two ways to approximate the posterior distribution: stochastic estimations and variational inference methods (Table 1).

Markov chain Monte Carlo (MCMC) methods are one of the most important stochastic approaches. MCMC methods provide an approximate unbiased estimator of the true estimator by sampling the posterior distribution according to the Markov process. MCMC methods can guarantee convergence to the true estimator with increasing sample size from the posterior, which can be computationally expensive, especially in BCNN [5, 11, 12, 24, 49]. On the other hand, many approaches have contributed to developing analytical approximations to the posterior distribution, especially in deep learning models. In this context, MacKay (1992) successfully applied the Laplace approximation to neural networks [42]. The Laplace approach attempts to estimate the posterior using a Gaussian distribution whose mean achieves the maximum a posteriori (MAP) and whose covariance matrix is the inverse of the Hessian matrix of the cost function used in the MAP estimate around this mean [54]. This approach focuses only on

Table 1 Summary of related works

Authors	Year	Method	Model	Datasets
Graves et al. [20]	2011	Stochastic variational inference	BRNN	TIMIT speech corpus
Ritter et al. [54]	2018	Laplace approximation	BNN	Mnist
Blundell et al. [8]	2015	Bayes by Backprop	BNN	Mnist
Yarin Gal et al. [16, 18]	2016	Monte Carlo Dropout	BCNN	Mnist, Cifar10
Kumar et al. [58]	2019	Bayes by Backprop	BCNN	Mnist, Cifar10, Cifar100
Jing Zhao et al. [70]	2020	Generalized Propagation Expectation	BNN (regression), BCNN (classification)	5 UCI datasets (BNN), Mnist (BCNN)
Pushkar Khairnar et al. [29]	2020	Bayes By Backprop	BCNN	Breast histopathological images

the properties of a single-mode MAP. In the case of multiple modes, they will produce different distributions, which often fail to approximate the posterior distribution, particularly in BCNN. Subsequently, many researchers have developed these approaches in deep learning models, using mainly variational inference methods (Hinton and Van Camp (1993) [26], Barber and Bishop (1998) [4], Graves (2011) [20], Blundell and al (Bayes by Backprop 2015) [8]. Again, in 2016, Graves attempted to approximate the gradient estimators for mixture models by employing the quantile functions as an alternate transform to the reparameterization trick [21]. In 2019, Kumar Shridhar applied the Bayes-by-backprop method using a unimodal distribution to model convolutional kernels [58].

Expectation Propagation (EP) is another method that estimates the true posterior by a simpler parametric factorized distribution from the exponential family. EP is based on minimizing the inverse form of the Kullback–Leibler (KL) divergence, $KL(p//q)$, rather than the direct form of $KL(q//p)$ used in variational inference (P.Minka [44]), (Hernandez-Lobato and P. Adams [25]). Sun et al. recently proposed "Generalized Expectation Propagation" (GEP) as an improved form of Expectation Propagation that can approximate multimodal posterior distributions, particularly in BCNN, by employing a mixture of exponential family distributions [61, 70]. Despite the scalability of the EP approach, the convergence using this method is not always assured, especially when using mixture models [13, 67]. On the other hand, Gal proved that training neural networks with dropout is similar to using a variational inference method with a Bernoulli distribution by transforming the dropout noise from the input space to the parameter space in neural networks (MC Dropout [16, 18]). While the MC Dropout method is suitable for deep learning models since it reduces overfitting and does not require as much modeling as other methods, it is also inflexible and cannot always fully express model uncertainty [10].

3 Background and preliminaries

3.1 Convolution neural network architecture

Assume we have N input images $x = \{(x_1, x_2, ..., x_N)\}$, and their labels $y = \{(y_1, y_2, ..., y_N)\}$, a standard CNN is a deep neural network that defines the output layer as a composition of convolution layers that extract the most significant features from the images, represented by, $c^{(i)}, i = 1, ..., N_c$, each $c^{(i)}$ is a convolution operation between the input $p^{(i-1)}$ and the filter matrix $k^{(i)}$ shifted by the bias $b_c^{(i)}$, followed by an activation function, $s_c^{(i)}, i = 1, ..., N_c$, and pooling layers, $p^{(i)}, i = 1, ..., N_c$, (N_c is the number of convolutional layers), each $p^{(i)}$ is a pooling function (which may be average-pooling or maximum-pooling). Finally, the fully connected layers, which are represented, after flattening (or vectorization), as a succession of hidden layers, $h^{(j)}, j = 1, ..., N_l$, (N_l is the number of hidden layers), each $h^{(j)}$ is a linear transformation,, accompanied by an activation function, $s_l^{(j)}, j = 1, ..., N_l$. The output $\hat{y}$ of CNN is represented as the last layer in the fully connected layers using Softmax probability as an activation function.

The parameter models of the CNN are $\theta = \{K, b_c, \omega, b_l\}$, where K, b_c are the kernels and the biases of the convolution layers, and ω, b_l are the weights and the biases of the fully connected hidden layers.

The standard CNN works to find a point estimate of the probable function f that relates the input x to the output y using the parameters models, in other words, it works to obtain a point approximation of the model parameters θ that fits the data-set $D = \{(x, y)\}$ well.

To achieve this point estimate, the model employs the backpropagation algorithm to minimize the cost function [9, 38, 57], which is proportional to maximizing the log-likelihood (ML), and occasionally with a regularization component included if the maximum a posteriori estimation (MAP) is used (See Algorithm 1).

Algorithm 1 Convolutional Neural Networks: Training Procedure

```
Input: $D = \{x, y\}$ (Dataset)
Initialization $\eta$ (learning rate), $\theta$ (parameters)
Parameter: $\theta$
Output: $\hat{\theta}$
 1: for m=1, Epochs do
 2:     for n=1, N do
 3:         Let $p^{(0)} = x_n$
            # Convolution layers
 4:         for n=1, $N_c$ do
 5:             $c^{(i)} = s_c^{(i)}(p^{(i-1)} * k^{(m)^{(i)}} + b_c^{(m)^{(i)}})$
 6:             $p^{(i)} = Pool(c^{(i)})$
 7:         end for
 8:         $h^{(0)} = Flattening(p^{(N_c)})$
            # Fully connected layers
 9:         for n=1,$N_l - 1$ do
10:             $h^{(j)} = s_l^{(j)}(\omega^{(m)^{(j)^T}} h^{(j-1)} + b_l^{(m)^{(j)}})$
11:         end for
            # Outputs layer
12:         $\hat{y}_n = f(x_n, K^{(m)}, b_c^{(m)}, \omega^{(m)}, b_l^{(m)})$
                $= Softmax\left(\omega^{(m)^{(N_l)^T}} h^{(N_l-1)} + b_l^{(m)^{(N_l)}}\right)$
13:     end for
14:     $\hat{y} = \{(\hat{y}_1, ...., \hat{y}_N)\}, f(x, \theta^{(m)}) = \left\{(f(x_1, \theta^{(m)}), ...., f(x_N, \theta^{(m)}))\right\}$
        # Define the loss function
15:     $Loss(\hat{y}, y) = Loss(f(x, \theta^{(m)}), y) = CrossEntropy(f(x, \theta^{(m)}), y)$
        # Backpropagation
16:     $\theta^{(m+1)} \longleftarrow \theta^{(m)} - \eta \frac{\partial Loss(f(x, \theta^{(m)}), y)}{\partial \theta^{(m)}}$
17: end for
```

Finally, this algorithm returns a single optimum parameter $\hat{\theta}$ that minimizes the cost function. For unseen input x^*, CNN uses the optimum parameter $\hat{\theta}$, to estimate their prediction.

$$\hat{\theta} = argmin\ Loss(y, \hat{y}) \tag{1}$$

$$\hat{y}^* = f(x^*, \hat{\theta}) = cnn(x^*, \hat{\theta}) \tag{2}$$

Where $\hat{\theta} = \{\hat{K}, \hat{b_c}, \hat{\omega}, \hat{b_l}\}$.

As we have shown, a standard CNN only gives a point estimate of the parameters, which works well on a large data set, but in some issues, larger quantities of data are not available. The problem with CNNs is that they quickly overfit with small data sets [17], which often results in overconfident predictions.

3.2 Variational inference

The Bayesian Convolutional Neural Network (BCNN) is a CNN trained by using Bayesian statistics, in which all parameters of the CNN are treated as stochastic components [17, 58].

The initial step in creating a BCNN is to determine the feedforward architecture of the CNN. We then apply the prior distribution $p(\theta)$ of the CNN weights θ, indicating our previous beliefs about the parameters. Next, we define the likelihood $p(y|x, \theta)$ as the independent conditional probability of the observed data given specific parameters. Generally, the likelihood for CNN models is defined as a categorical distribution of the softmax probabilities, as shown below:

$$\begin{aligned} p(y|x, \theta) &= \prod_{n=1}^{N} p(y_n|x_n, \theta) = \prod_{n=1}^{N} p(y_n|f(x_n, \theta)) \\ &= \prod_{n=1}^{N} Categorical(f^1(x_n, \theta), ..., f^C(x_n, \theta)) \end{aligned} \tag{3}$$

Where $f^c(x_n, \theta) = Softmax\left(\omega_c^{(n_l)^T} h^{(n_l-1)} + b_{l_c}^{(n_l)}\right) = \frac{exp\left(\omega_c^{(n_l)^T} h^{(n_l-1)} + b_{l_c}^{(n_l)}\right)}{\Sigma_{c'}^{C} exp\left(\omega_{c'}^{(n_l)^T} h^{(n_l-1)} + b_{l_{c'}}^{(n_l)}\right)}$,

and C is the number of output classes.

We can then obtain the posterior distribution of CNN parameters given the observed data $p(\theta|x, y)$ using the Bayes theorem as follows:

$$\begin{aligned} p(\theta|x, y) &= \frac{p(y|x, \theta)p(\theta)}{p(D)} = \frac{p(y|x, \theta)p(\theta)}{\int_\theta p(y|x, \theta')p(\theta')d\theta'} \\ &\propto p(y|x, \theta)p(\theta) \end{aligned} \tag{4}$$

Deep learning models are made of a huge number of parameters, which makes determining the posterior distribution $p(\theta|x, y)$ tricky since computing the evidence term $\int_\theta p(y|x, \theta')p(\theta')d\theta'$ is hard.

The Variational Inference (VI) method was developed for solving an optimization problem to approximate the posterior distribution $p(\theta|x, y)$ with a simple parametric distribution $q_\phi(\theta)$ to overcome this problem [7, 69].

VI looks for an optimal variational parameter $\hat{\phi}$ such that the variational distribution $q_{\hat{\phi}}(\theta)$ is as close as possible to the true posterior $p(\theta|x, y)$ based on the Kullback–Leibler divergence [35], which is expressed as follows:

$$KL(q_\phi(\theta)||p(\theta|x, y)) = \int_\theta q_\phi(\theta) \log\left(\frac{q_\phi(\theta)}{p(\theta|x, y)}\right) d\theta \tag{5}$$

To compute the $KL(q\|p)$, you must first compute the posterior distribution. As a result, the problem still exists. To get around this, we use the Evidence Lower BOund (ELBO) function $\mathscr{L}$, which can be obtained from $KL(q\|p)$ and the Bayes Formula as follows:

$$KL(q_\phi(\theta)||p(\theta|x,y)) = \log p(D) - \mathscr{L}(\phi) \tag{6}$$

Where

$$\begin{aligned}\mathscr{L}(\phi) &= \int_\theta q_\phi(\theta)\log p(y|x,\theta)d\theta \\ &\quad - \int_\theta q_\phi(\theta)\log\left(\frac{q_\phi(\theta)}{p(\theta)}\right)d\theta \\ &= E_{q_\phi(\theta)}[\log p(y|x,\theta)] - KL(q_\phi(\theta)||p(\theta))\end{aligned} \tag{7}$$

Minimizing the KL divergence is now equivalent to maximizing the ELBO function $\mathscr{L}$ since the $\log p(D)$ is constant over the variational parameters.

ELBO maximization needs to maximize the first term of the last equation, which denotes the expected log-likelihood and minimize the second term, indicating the KL divergence between $q_\phi(\theta)$ and $p(\theta)$. Generally, the second term serves as a regularizer [16].

The prediction in BCNN of a new input x^* is a probability distribution $p(y^*|x^*, D)$, called the predictive distribution [66]. It is defined by the expectation over the posterior distribution of the model's output, as shown below:

$$\begin{aligned}p(y^*|x^*,x,y) &= \int_\theta p(y^*|x^*,\theta)p(\theta|x,y)d\theta \\ &= E_{p(\theta|x,y)}[p(y^*|x^*,\theta)]\end{aligned} \tag{8}$$

Using the variational inference method, the predictive distribution can be approximated as follows:

$$\begin{aligned}p(y^*|x^*,x,y) &\approx \int_\theta p(y^*|x^*,\theta)q_{\hat\phi}(\theta)d\theta \\ &= E_{q_{\hat\phi}(\theta)}[p(y^*|x^*,\theta)] \\ &= q(y^*|x^*)\end{aligned} \tag{9}$$

Where $\hat\phi$ are the optimal variational parameters.

3.3 Bayes by backprop

Variational inference is a powerful statistical estimate for Bayesian inference. However, the stochasticity of the parameters prevents back-propagation from working in deep learning models. To overcome this issue, Blundell et al. proposed the Bayes-by-backprop algorithm [8]. Bayes by Backprop is indeed a variational inference technique that looks for the variational parameters $\hat\phi$ that minimize the KL divergence between the posterior distribution $p(\theta|x,y)$ and the variational distribution $q_\phi(\theta)$.

$$\begin{aligned}\hat\phi &= \arg\min_\phi KL(q_\phi(\theta)||p(\theta|x,y)) \\ &= \arg\min_\phi KL(q_\phi(\theta)||p(\theta)) - E_{q_\phi(\theta)}[\log p(y|x,\theta)] \\ &= \arg\min_\phi \underbrace{\int_\theta q_\phi(\theta)\Big(\underbrace{\log q_\phi(\theta) - \log p(\theta) - \log p(y|x,\theta)}_{l(\theta,\phi)}\Big)d\theta}_{\mathscr{L}}(\mathscr{D},\phi)\end{aligned} \tag{10}$$

Where $\mathscr{L}(\mathscr{D},\phi)$ is the negative ELBO function.

The new aspect of the Bayes-by-backprop algorithm is to apply the reparametrization trick technique to the model parameters [30, 31]. The idea is to transform the randomness of the model parameters, θ, which is simulated from a parametric distribution $q_\phi(\theta)$, to another random variable ϵ, that follows a non-parametric distribution, $q(\epsilon)$, by reparameterizing θ as a deterministic and differentiable function of the variational parameters ϕ and ϵ, $g(\phi,\epsilon)$ such that $\theta = g(\phi,\epsilon)$. Therefore, we may compute the gradients $\nabla_\theta l(\theta,\phi)$ by backpropagating via θ, which is now non-stochastic (See Algorithm 2).

As a result, Blundell et al. [8] proposed that if $q_\phi(\theta)d\theta = q(\epsilon)d\epsilon$, and for a differentiable function $l(\theta,\phi)$, we get:

$$\begin{aligned}\frac{\partial}{\partial\phi}\mathscr{L}(\mathscr{D},\phi) &= \frac{\partial}{\partial\phi}E_{q_\phi(\Theta)}\Big[l(\theta,\phi)\Big] \\ &= E_{q(\epsilon)}\left[\frac{\partial l(\theta,\phi)}{\partial\theta}\frac{\partial\theta}{\partial\phi} + \frac{\partial l(\theta,\phi)}{\partial\phi}\right]\end{aligned} \tag{11}$$

Where $\frac{\partial\theta}{\partial\phi} = \frac{\partial g(\phi,\epsilon)}{\partial\phi}$ in the last expression. For more details about the last formula, see [8].

As $\frac{\partial}{\partial\phi}\mathscr{L}(\mathscr{D},\phi)$ is also hard to compute, we can use Monte Carlo sampling to estimate it. Using the reparameterization trick, we first sample ϵ from the non-parametric distribution $q(\epsilon)$ and then apply the deterministic function g, such that $\theta = g(\phi,\epsilon) \sim q_\phi(\theta)$.

As a result, we can approximate $\frac{\partial}{\partial\phi}\mathscr{L}(\mathscr{D},\phi)$ as follows:

$$\begin{aligned}\frac{\partial}{\partial\phi}\mathscr{L}(\mathscr{D},\phi) &\approx \frac{\partial}{\partial\phi}\hat{\mathscr{L}}(\theta,\phi) \\ &= \frac{1}{T}\sum_{t=1}^{T}\Bigg[\frac{\partial l(g(\phi,\epsilon^{(t)}),\phi)}{\partial\theta^{(t)}}\frac{\partial g(\phi,\epsilon^{(t)})}{\partial\phi} \\ &\quad + \frac{\partial l(g(\phi,\epsilon^{(t)}),\phi)}{\partial\phi}\Bigg]\end{aligned} \tag{12}$$

Where $l(g(\phi, \epsilon^{(t)}), \phi) = \log q_\phi(g(\phi, \epsilon^{(t)})) - \log p(g(\phi, \epsilon^{(t)})) - \log p(y|x, g(\phi, \epsilon^{(t)}))$, $\epsilon^{(t)} \sim q(\epsilon)$, T is the number of samples, and $\frac{\partial}{\partial\phi}\hat{\mathscr{L}}(\theta, \phi)$ is an unbiased estimator of $\frac{\partial}{\partial\phi}\mathscr{L}(\mathscr{D}, \phi)(E_{q(\epsilon)}\left[\frac{\partial}{\partial\phi}\hat{\mathscr{L}}(\theta, \phi)\right] = \frac{\partial}{\partial\phi}\mathscr{L}(\mathscr{D}, \phi))$.

Algorithm 2 Bayes by Backprop Algorithm [8]

```
Input: $D = \{x, y\}$ (Dataset)
Initialization: $\eta$ (learning rate), $\phi$
Parameter: $\phi$ (variational parameters)
Hyperparameters: Epochs, T (number of MC samples)
Output: $\hat{\phi}$

1:  for m=1, Epochs do
2:    for t=1, T do
3:      Sample $\varepsilon^{(t)} \sim q(\varepsilon)$
4:      Let $\theta^{(t)} = g(\phi^m, \varepsilon^{(t)})$ # Reparameterization trick
5:      $l(\theta^{(t)}, \phi^m) = \log q_{\phi^m}(\theta^{(t)}) - \log p(\theta^{(t)}) - \log p(y|x, \theta^{(t)})$
6:      $\frac{\partial}{\partial\phi} l(\theta^{(t)}, \phi^m) = \frac{\partial l(\theta^{(t)}, \phi^m)}{\partial\theta^{(t)}} \frac{\partial\theta^{(t)}}{\partial\phi} + \frac{\partial l(\theta^{(t)}, \phi^m)}{\partial\phi}$
7:    end for
      # Compute the gradient of the negative ELBO function defined in Eq. 12
8:    $\frac{\partial}{\partial\phi}\mathscr{L}(\theta, \phi^m) = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial}{\partial\phi} l(\theta^{(t)}, \phi^m)$
      # Backpropagation over the variational parameters
9:    $\phi^{m+1} \longleftarrow \phi^m - \eta\frac{\partial}{\partial\phi}\mathscr{L}(\theta, \phi^m)$
10: end for
```

3.4 Gaussian mixture model

A Gaussian Mixture Model (GMM) is a probability distribution defined as a linear convex combination of Gaussian distributions [13, 53]. Therefore, we can express a GMM composed of a K-component Gaussian density as follows:

$$p_\phi(\theta) = \sum_{k=1}^{K} \pi_k \mathrm{N}(\theta|\mu_k, \Sigma_k) \tag{13}$$

Where $\boldsymbol{\theta}$ is a D-dimensional vector, $\{\pi_k, k = 1, 2, ..., K\}$ are the mixture weights that satisfy the constraint that $\sum_{k=1}^{K} \pi_k = 1, with \quad 0 < \pi_k < 1$, and $\{\mathrm{N}(\theta|\mu_k, \Sigma_k), k = 1, 2, ..., K\}$ are the components distributions. Each component is a multivariate Gaussian distribution of the following form:

$$\mathrm{N}(\theta|\mu_k, \Sigma_k) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(\theta - \mu_k)^T \Sigma_k^{-1}(\theta - \mu_k)\right) \tag{14}$$

Where μ_k is a d-dimensional mean vector, Σ_k is a d×d-dimensional covariance matrix of the corresponding Gaussian distribution. $\phi = \{(\pi_k, \phi_k = \{\mu_k, \Sigma_k\}), k = 1, 2, ..., K\}$ represents the full parameters of the Gaussian mixture model $p_\phi(\theta)$.

For various theoretical and computational reasons, the Gaussian distribution is the most preferred unimodal distribution in real-world modelling. However, some challenging applications, such as image classification, require more sophisticated distributions to model them.

As a result, using a unimodal model in this situation is frequently ineffective [53],

[13]. To solve this issue, we can represent these applications using a GMM, which is a combination of several unimodal Gaussian distributions that can provide more statistical information about the problem than single-mode distributions.

4 Bayes by backprop using mixture models

A mixture model is a powerful tool that may be used in variational inference as an approximated posterior distribution. However, reparameterizing the parameters of mixture models is challenging since they combine component distributions using discrete-categorical variables

$k \sim Cat(\{\pi_k\}_{k=1}^{K})$. To address this problem, Roeder et al. propose that the expectation over the mixture model be computed by taking the sum of the mixture weights outside the expectation and then sampling equally from each component distribution, [47, 55].

As a result, if each component distribution is reparameterizable, we may reparameterize the mixture models.

Let $q_\phi(\theta) = \sum_{k=1}^{K} \pi_k q_{\phi_k}(\theta)$ as a mixture distribution made up of K component distributions $q_{\phi_k}(\theta)$, combined by K mixture weights $\{\pi_k, k = 1, 2, ..., K\}$.

If $q_{\phi_k}(\theta_k)d\theta_k = q(\xi_k)d\xi_k$, and $\theta_k = g(\phi_k, \xi_k)$, for $k = 1, 2, ..., K$, with g is a differentiable function, we can approximate the expectation $E_{q_\phi(\theta)}[f(\theta)]$, as follows:

$$\begin{aligned} E_{q_\phi(\theta)}\left[f(\theta)\right] &= \sum_{k=1}^{K} \pi_k E_{q(\xi_k)}\left[f(g(\phi_k, \xi_k))\right] \\ &\approx \sum_{k=1}^{K} \frac{\pi_k}{T} \sum_{t=1}^{T} f(g(\phi_k, \xi_k^{(t)})) \end{aligned} \tag{15}$$

Where $\xi_k^{(t)} \sim q(\xi_k)$, and T is the number of samples.

Proof:

$$E_{q_\phi(\theta)}\Big[f(\theta)\Big] = \int_\theta \sum_{k=1}^{K} \pi_k q_{\phi_k}(\theta) f(\theta) d\theta$$

$$= \sum_{k=1}^{K} \pi_k \int_\theta q_{\phi_k}(\theta) f(\theta) d\theta$$

$$= \sum_{k=1}^{K} \pi_k \int_{\theta_k} q_{\phi_k}(\theta_k) f(\theta_k) d\theta_k$$

(Linearity of integral)

$$= \sum_{k=1}^{K} \pi_k \int_{\xi_k} q(\xi_k) f(g(\phi_k, \xi_k)) d\xi_k$$

Reparameterization the parameters of each component distribution as $\theta_k = g(\phi_k, \xi_k)$

$$= \sum_{k=1}^{K} \pi_k E_{q(\xi_k)}[f(g(\phi_k, \xi_k))]$$

$$\approx \sum_{k=1}^{K} \frac{\pi_k}{T} \sum_{t=1}^{T} f(g(\phi_k, \xi_k^{(t)}))$$

□

With $\xi_k^{(t)} \sim q(\xi_k)$, for $k = \{1, 2, ..., K\}$.

The latter expression is the result of applying Monte Carlo sampling to each component distribution $q(\xi_k)$, taking into account the independence of ξ_k, resulting from the independence of parameters θ_k, for $k = \{1, 2, ..., K\}$.

Furthermore, using this approximation, we obtain an unbiased estimator as shown below:

$$E_{\prod_{k=1}^{K} q(\xi_k)}\Big[\sum_{k=1}^{K} \frac{\pi_k}{T} \sum_{t=1}^{T} f(g(\phi_k, \xi_k^{(t)}))\Big]$$

$$= \sum_{k=1}^{K} \pi_k E_{q(\xi_k)}\Big[\frac{1}{T} \sum_{t=1}^{T} f(g(\phi_k, \xi_k^{(t)}))\Big]$$

$$= \sum_{k=1}^{K} \frac{\pi_k}{T} \sum_{t=1}^{T} E_{q(\xi_k)}\Big[f(g(\phi_k, \xi_k^{(t)}))\Big]$$

$$= \sum_{k=1}^{K} \pi_k E_{q(\xi_k)}\Big[f(g(\phi_k, \xi_k))\Big]$$

$$= \sum_{k=1}^{K} \pi_k E_{q_{\phi_k}(\theta_k)}\Big[f(\theta_k)\Big]$$

$$= E_{q_\phi(\theta)}\Big[f(\theta)\Big]$$

Instead of sampling a discrete random variable $k \sim Cat(\{\pi_k\}_{k=1}^{K})$ and then sampling θ from the associated component distribution $q_{\phi_k}(\theta)$, the approach described above takes samples from each component distribution equally and then combines them using the mixture weights. Although the last method is more computationally expensive than the first since it requires K-implementations of the function f to obtain one Monte Carlo estimate of the expectation

$E_{q_\phi(\theta)}[f(\theta)]$, it allows us to reparameterize the parameters of the mixture model and also gives a differentiable estimate (See Fig. 2), which is not available in the first approach [15, 40].

Therefore, if all conditions are satisfied, we can apply the Bayes by Backprop method to the differentiable and continuous function l (Eq. 10), using a mixture model as an approximate distribution (Algorithm 3), as illustrated below:

for $j = 1, 2, ..., K$:

$$\frac{\partial}{\partial \phi_j} \mathscr{L}(D, \phi) = \frac{\partial}{\partial \phi_j} \mathscr{L}(D, (\phi_1, .., \phi_j, .., \phi_K))$$

$$= \frac{\partial}{\partial \phi_j} E_{q_\phi(\theta)}\Big[l(\theta, \phi)\Big]$$

$$= \frac{\partial}{\partial \phi_j} \sum_{k=1}^{K} \pi_k E_{q_{\phi_k}(\theta)}\Big[l(\theta, \phi)\Big]$$

$$= \frac{\partial}{\partial \phi_j} \sum_{k=1}^{K} \pi_k E_{q(\xi_k)}\Big[l(g(\phi_k, \xi_k), \phi)\Big]$$

$$= \sum_{k=1}^{K} \pi_k E_{q(\xi_k)}\Big[\frac{\partial l(g(\phi_k, \xi_k), \phi)}{\partial \theta_k} \frac{\partial g(\phi_k, \xi_k)}{\partial \phi_j} + \frac{\partial l(g(\phi_k, \xi_k), \phi)}{\partial \phi_j}\Big]$$

Where $q_\phi(\theta) = \sum_{k=1}^{K} \pi_k q_{\phi_k}(\theta)$ is a mixture model,

$l(\theta, \phi) = \log q_\phi(\theta) - \log p(\theta) - \log p(y|x, \theta)$, and

$q_{\phi_k}(\theta_k) d\theta_k = q(\xi_k) d\xi_k$, for $k = 1, 2, ..., K$.

Since $\frac{\partial}{\partial \phi_j} \mathscr{L}(\phi, D)$ is also computationally intricate, we can estimate it using Eq. 15, as follows:

for $j = 1, 2, ..., K$:

$$\frac{\partial}{\partial \phi_j} \mathscr{L}(D, \phi) \approx \frac{\partial}{\partial \phi_j} \hat{\mathscr{L}}(\boldsymbol{\theta}, \boldsymbol{\phi})$$

$$= \frac{1}{T} \sum_{k=1}^{K} \sum_{t=1}^{T} \pi_k \frac{\partial l(\theta_k^{(t)}, \phi)}{\partial \theta_k^{(t)}} \frac{\partial g(\phi_k, \xi_k^{(t)})}{\partial \phi_j} + \frac{1}{T} \sum_{k=1}^{K} \sum_{t=1}^{T} \pi_k \frac{\partial l(g(\phi_k, \xi_k^{(t)}), \phi)}{\partial \phi_j} \quad (16)$$

Where $\xi_k^{(t)} \sim q(\xi_k)$ for $t = 1, 2, ..., T$, and $k = 1, 2, ..., K$,

$l(g(\phi_k, \epsilon^{(t)}), \phi) = \log \Big(\sum_{i=1}^{K} \pi_i q_{\phi_i}(g(\phi_k, \xi_k^{(t)})) \Big)$

$- \log p(g(\phi_k, \xi_k^{(t)})) - \log p(y|x, g(\phi_k, \xi_k^{(t)}))$, and

$\frac{\partial}{\partial \phi_j}\hat{\mathcal{L}}(\theta, \phi)$ is an unbiased estimator of $\frac{\partial}{\partial \phi_j}\mathcal{L}(\mathcal{D}, \phi)$. $\left(E_{\prod_{k=1}^{K} q(\xi_k)}\left[\frac{\partial}{\partial \phi_j}\hat{\mathcal{L}}(\theta, \phi)\right] = \frac{\partial}{\partial \phi_j}\mathcal{L}(\mathcal{D}, \phi), \text{ for } j = 1, 2, ..., K\right)$.

Algorithm 3 Bayes by Backprop using mixture model

```
Input: $D = \{x, y\}$ (Dataset)
Initialization $\eta$ (learning rate), $\phi_1, ..., \phi_K$ ($K$: number of components)
Parameter: $\phi_1, ..., \phi_K$ (variational parameters)
Hyperparameters: $\pi = \{\pi_1, ..., \pi_K\}$ (mixture weights), Epochs, $T$ (number of samples)
Output: $\hat{\phi}_1, ..., \hat{\phi}_K$

1:  for m=1, Epochs do
2:    for t=1, $T$ do
3:      for k=1, $K$ do
4:        Sample $\xi_k^{(t)} \sim q(\xi_k)$
          # Reparameterize $q_{\phi_k}(\theta)$
5:        Let $\theta_k^{(t)} = g(\phi_k^m, \xi_k^{(t)})$
6:        $l(\theta_k^{(t)}, \phi^m) = \log\left(\sum_{i=1}^{K} \pi_i q_{\phi_i^m}(\theta_k^{(t)})\right) - \log p(\theta_k^{(t)}) - \log p(y|x, \theta_k^{(t)})$
7:        $\frac{\partial}{\partial \phi_j} l(\theta_k^{(t)}, \phi^m) = \frac{\partial l(\theta_k^{(t)}, \phi^m)}{\partial \theta_k^{(t)}} \frac{\partial \theta_k^{(t)}}{\partial \phi_j} + \frac{\partial l(\theta_k^{(t)}, \phi_k^m)}{\partial \phi_j}$
8:      end for
9:    end for
      # Compute the gradient of the N-Elbo w.r.t to $\phi_j$ as defined in Eq. 16
10:   $\frac{\partial}{\partial \phi_j}\hat{\mathcal{L}}(\theta, \phi^m) = \frac{1}{T}\sum_{k=1}^{K}\sum_{t=1}^{T} \pi_k \frac{\partial}{\partial \phi_j} l(\theta_k^{(t)}, \phi^m)$, for j= 1,...,K
      # Backpropagation over $\phi_j$
11:   $\phi_j^{m+1} \longleftarrow \phi_j^m - \eta \frac{\partial}{\partial \phi_j}\hat{\mathcal{L}}(\theta, \phi^m)$, for j= 1,...,K
12: end for
```

4.1 Bayesian convolution neural network with GMM

In this section, we will try to apply the Bayes By Backprop method to convolutional neural networks using a Gaussian mixture model (BBGMM) as an approximate distribution of the true posterior, and we will show how to construct, train, and evaluate BCNN using this distribution.

A convolutional neural network is a deep learning model characterized by two basic steps. First, we extract the most significant features of inputs using kernels in convolutional layers, and second, we classify the inputs using fully connected layers and a softmax function in the output layer (See Fig. 1).

As a result, the CNN parameters are expressed as follows: $\theta = \{F, b_c, \omega, b_l\}$, where $F = \{\{f_{h_{i,p}, w_{i,p}, c_{i,p}}\}_{i,p=1}^{N_c, p_i}\}, b_c = \{\{b_{c_i}\}_{i=1}^{i=N_c}\}$ are the kernels and the biases of the convolutional layers, and $\omega = \{\{W_j\}_{j=1}^{j=N_l}\}$, $b_l = \{\{b_{l_j}\}_{j=1}^{j=N_l}\}$ are the weights and the biases of the fully connected layers.

Following the Bayesian approach, these parameters are represented as stochastic kernels in the convolution layers and as stochastic matrices in the fully connected layers. Before seeing the data, the Bayes by Backprop algorithm, like the other variational inference methods, requires setting a prior distribution for all CNN parameters $p(\theta) = p(F, b_c, \omega, b_l)$ as prior beliefs about the possible parameters that fit the data. After seeing the data $D = \{(x, y)\}$, we have to determine the model's probability $p(y|x, \theta)$ (the likelihood) for the outputs $y = \{(y_1, y_2, ..., y_N)\}$ given the inputs $x = \{(x_1, x_2, ..., x_N)\} \in \mathcal{R}^{HxWxCxN}$ and parameters $\theta = \{F, b_c, \omega, b_l\}$.

Then, we use a mixture of K fully-factorized normal distributions (i.e., with diagonal covariances) as a variational distribution of the CNN parameters, as shown below:

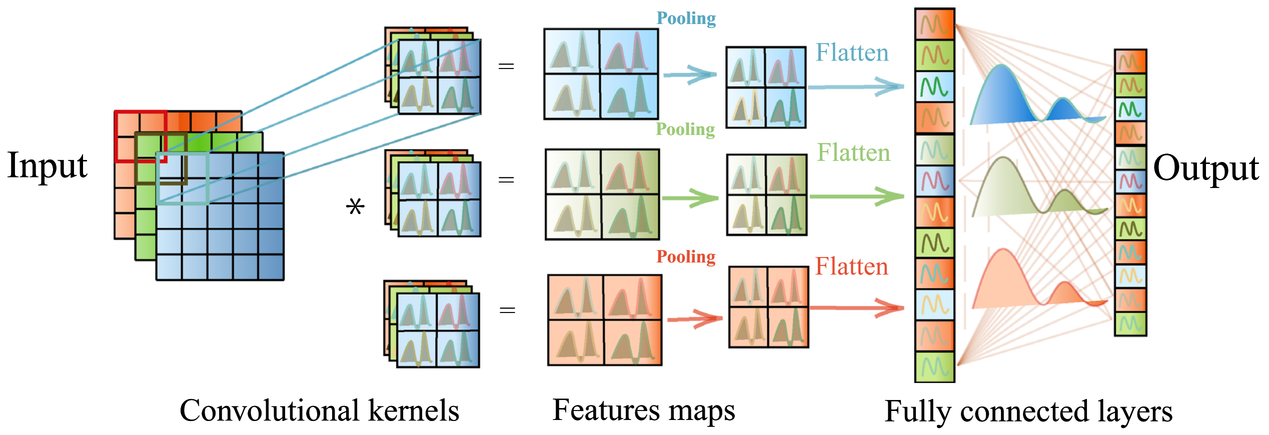


Fig. 1 Bayesian convolution neural network

$$
\begin{aligned}
q_{\phi_k}(\theta) &= \sum_{k=1}^{K} \pi_k q_{\phi_k}(F, b_c, \omega, b_l) \\
&= \sum_{k=1}^{K} \pi_k \underbrace{q_{\phi_{k_F}}(F) q_{\phi_{k_{bc}}}(b_c)}_{Convolution-layers} \underbrace{q_{\phi_{k_\omega}}(\omega) q_{\phi_{k_{bl}}}(b_l)}_{Fully-Connected-layers}
\end{aligned} \tag{17}
$$

Where

$$q_{\phi_{k_F}}(F) = \prod_{i,p,h,w,c=1}^{N_c,p_i,h_i,w_i,c_i} \mathrm{N}\left(f_{i,p,h,w,c} | \mu_{k_{f_{i,p,h,w,c}}}, \sigma^2_{k_{f_{i,p,h,w}}}\right)$$

$$q_{\phi_{k_{bc}}}(bc) = \prod_{i,p=1}^{N_c,p_i} \mathrm{N}\left(bc_{i,p} | \mu_{k_{bc_{i,p}}}, \sigma^2_{k_{bc_{i,p}}}\right)$$

$$q_{\phi_{k_\omega}}(\omega) = \prod_{j,m,n=1}^{N_l,L_j,L_{j+1}} \mathrm{N}\left(w_{j,m,n} | \mu_{k_{w_{j,m,n}}}, \sigma^2_{k_{w_{j,m,n}}}\right)$$

$$q_{\phi_{k_{bl}}}(bl) = \prod_{j,n=1}^{N_l,L_{j+1}} \mathrm{N}\left(bl_{j,n} | \mu_{k_{bl_{j,n}}}, \sigma^2_{k_{bl_{j,n}}}\right)$$

N_c is the number of convolutional layers, p_i, h_i, w_i, and c_i represent, respectively, the number, width, height, and the channels number of kernels in the i^{th} convolutional layer, N_l is the number of fully connected layers, L_j, and L_{j+1} denote the number of neurons in the j^{th} and the $j+1^{th}$ layers, respectively.

The Gaussian distribution $\mathrm{N}(\theta|\mu, \sigma^2)$ can be reparameterized to the unit Gaussian $\mathrm{N}(\epsilon|0, I)$ by using a differential transform of the parameters expressed as follows: $\theta = \mu + \sigma \odot \epsilon$, where $\odot$ is an element wise product, and ϵ is a free-noise parameter of the unit Gaussian. To prevent receiving negative values for σ, we can rewrite them as follows: $\sigma = \log(1 + \exp(\sigma))$. As a result, we can reparameterize the parameters of the Gaussian mixture model by reparameterizing them over each Gaussian distribution component (See Fig. 2), as shown below:

for $k = 1, ..., K$.

$$\theta_k = g(\phi_k, \epsilon_k) = \mu_k + \log(1 + \exp(\sigma_k)) \odot \epsilon_k \tag{18}$$

With $\epsilon_k \sim \mathrm{N}(\epsilon|0, I)$, and $\phi_k = \{(\mu_k, \sigma_k)\}$.

This allows applying the Bayes by backprop algorithm using GMM to deep neural networks, including CNNs (see Algorithm 4), as follows:

for $j = 1, ..., K$.

$$
\begin{aligned}
\frac{\partial}{\partial \phi_j} \mathscr{L}(\mathscr{D}, \phi) &= \sum_{k=1}^{K} \pi_k E_{\mathrm{N}(\epsilon_k|0,I)} \Big[\frac{\partial l(\boldsymbol{\theta_k}, \boldsymbol{\phi})}{\partial \theta_k} \frac{\partial g(\phi_k, \epsilon_k)}{\partial \phi_j} \\
&\quad + \frac{\partial l(g(\phi_k, \epsilon_k), \phi)}{\partial \phi_j} \Big] \\
&\approx \frac{1}{T} \sum_{t=1}^{T} \sum_{k=1}^{K} \pi_k \Big[\frac{\partial l(\theta_k^{(t)}, \phi)}{\partial \theta_k^{(t)}} \frac{\partial g(\phi_k, \epsilon_k^{(t)})}{\partial \phi_j} \\
&\quad + \frac{\partial l(g(\phi_k, \epsilon_k^{(t)}), \phi)}{\partial \phi_j} \Big] \\
&= \frac{\partial}{\partial \phi_j} \hat{\mathscr{L}}(\theta, \phi)
\end{aligned} \tag{19}
$$

$$
\begin{cases}
\frac{\partial}{\partial \mu_j} \mathscr{L}(D, \phi) \approx \frac{1}{T} \sum_{t=1}^{T} \Big[\sum_{k=1}^{K} \pi_k \frac{\partial l(g(\phi_k, \epsilon_k^{(t)}), \phi)}{\partial \mu_j} \\
\quad + \pi_j \frac{\partial l(\theta_j^{(t)}, \phi)}{\partial \theta_j^{(t)}} \Big] \\
\quad = \frac{\partial}{\partial \mu_j} \hat{\mathscr{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}) \\
\frac{\partial}{\partial \sigma_j} \mathscr{L}(D, \phi) \approx \frac{1}{T} \sum_{t=1}^{T} \Big[\sum_{k=1}^{K} \pi_k \frac{\partial l(g(\phi_k, \epsilon_k^{(t)}), \phi)}{\partial \sigma_j} \\
\quad + \pi_j \frac{\partial l(\theta_j^{(t)}, \phi)}{\partial \theta_j^{(t)}} \frac{\epsilon_j^{(t)}}{1 + \exp(-\sigma_j)} \Big] \\
\quad = \frac{\partial}{\partial \sigma_j} \hat{\mathscr{L}}(\theta, \phi)
\end{cases}
$$

Where, $\epsilon_k^{(t)} \sim \mathrm{N}(\epsilon_k|0, I)$, for $k = 1, ..., K$, and $t = 1, ..., T$,

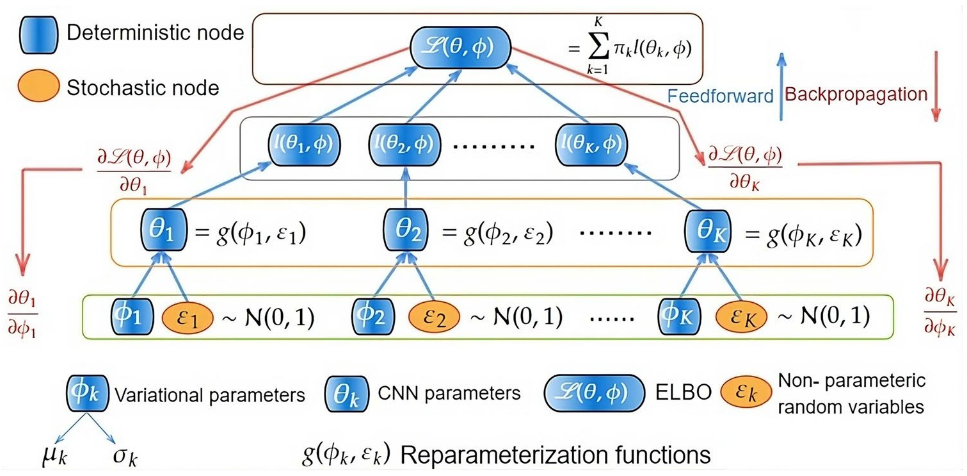

Deterministic node
Stochastic node
$\mathscr{L}(\theta,\phi)$
$= \sum_{k=1}^{K} \pi_k l(\theta_k, \phi)$
Feedforward
Backpropagation
$\frac{\partial \mathscr{L}(\theta,\phi)}{\partial \theta_1}$
$\frac{\partial \mathscr{L}(\theta,\phi)}{\partial \theta_K}$
$l(\theta_1,\phi)$ $l(\theta_2,\phi)$ .... $l(\theta_K,\phi)$
$\theta_1 = g(\phi_1, \varepsilon_1)$ $\theta_2 = g(\phi_2, \varepsilon_2)$ .... $\theta_K = g(\phi_K, \varepsilon_K)$
$\frac{\partial \theta_1}{\partial \phi_1}$
$\frac{\partial \theta_K}{\partial \phi_K}$
ϕ_1 $\varepsilon_1 \sim N(0,1)$ ϕ_2 $\varepsilon_2 \sim N(0,1)$ ...... ϕ_K $\varepsilon_K \sim N(0,1)$
ϕ_k Variational parameters
θ_k CNN parameters
$\mathscr{L}(\theta,\phi)$ ELBO
ε_k Non- parametric random variables
μ_k σ_k
$g(\phi_k, \varepsilon_k)$ Reparameterization functions

Fig. 2 Reparameterization trick on GMM using K Gaussian Components

and $l(\theta_k, \phi) = \log\left(\sum_{i=1}^{K} \pi_i \mathrm{N}(\theta_k|\mu_i, \sigma_i^2)\right) - \log p(\theta_k) - \log p(y|x, \theta_k)$.

Algorithm 4 Training BCNN using Gaussian mixture model (BBGMM)

Input: $D = \{x, y\}$ (Dataset)
Initialization η (learning rate), $\mu = \{\mu_1, ..., \mu_K\}$, $\sigma = \{\sigma_1, ..., \sigma_K\}$
Parameter: μ (means),σ (standard deviations)
Hyperparameters: $\pi = \{\pi_1, ..., \pi_K\}$(mixture weights), Epochs, T (number of samples)
Output: $\hat{\mu}, \hat{\sigma}$

```
1: for m=1, Epochs do
2:   for t=1, $T$ do
3:     for k=1, $K$ do
4:       Sample $\varepsilon_k^{(t)} \sim \mathrm{N}(\varepsilon_k|0, I)$
         # Reparameterize $N(\theta_k|\mu_k, \sigma^2)$
5:       Let $\theta_k^{(t)} = \mu_k^m + \log(1 + \exp(\sigma_k^m)) \odot \varepsilon_k^{(t)}$
         # Define the likelihood on $\theta_k^{(t)}$
6:       $p(y|x, \theta_k^{(t)}) = p(y|\mathrm{cnn}(x, \theta_k^{(t)}))$
7:       $l(\theta_k^{(t)}, \phi^m) = \log\left(\sum_{i=1}^{K} \pi_i \mathrm{N}(\theta_k^{(t)}|\mu_i^m, \sigma_i^{m^2})\right) - \log p(\theta_k^{(t)}) - \log p(y|x, \theta_k^{(t)})$
8:     end for
9:   end for
     # Compute the gradient of the N-Elbo w.r.t to $\mu_j$ as defined in Eqs. 19
10:  $\frac{\partial}{\partial \mu_j} \hat{\mathscr{L}}(\theta, \phi^m) = \frac{1}{T} \sum_{t=1}^{T} \left[ \sum_{k=1}^{K} \pi_k \frac{\partial l(g(\phi_k^m, \varepsilon_k^{(t)}), \phi^m)}{\partial \mu_j} + \pi_j \frac{\partial l(\theta_j^{(t)}, \phi^m)}{\partial \theta_j^{(t)}} \right]$
     # Compute the gradient of the N-Elbo w.r.t to $\sigma_j$ as defined in Eqs. 19
11:  $\frac{\partial}{\partial \sigma_j} \hat{\mathscr{L}}(\theta, \phi^m) = \frac{1}{T} \sum_{t=1}^{T} \sum_{k=1}^{K} \pi_k \frac{\partial l(g(\phi_k^m, \varepsilon_k^{(t)}), \phi^m)}{\partial \sigma_j} + \frac{\pi_j}{T} \sum_{t=1}^{T} \frac{\partial l(\theta_j^{(t)}, \phi^m)}{\partial \theta_j^{(t)}} \frac{\varepsilon_j^{(t)}}{1 + \exp\left(-\sigma_j^m\right)}$
     # Backpropagation over $\mu_j$
12:  $\mu_j^{m+1} \longleftarrow \mu_j^m - \eta \frac{\partial}{\partial \mu_j} \hat{\mathscr{L}}(\theta, \phi^m)$, for j= 1,...,K
     # Backpropagation over $\sigma_j$
13:  $\sigma_j^{m+1} \longleftarrow \sigma_j^m - \eta \frac{\partial}{\partial \sigma_j} \hat{\mathscr{L}}(\theta, \phi^m)$, for j= 1,...,K
14: end for
```

Having obtained the optimal variational distribution, we can use it to estimate the predictive distribution $p(y^*|x^*, D)$ for an unseen data input x^* using the observed data $D = \{(x, y)\}$, as follows:

$$\begin{aligned} p(y^*|x^*, x, y) &\approx \int q_{\hat{\phi}}(\theta) p(y^*|x^*, \theta) d\theta \\ &= \int \left(\sum_{k=1}^{K} \pi_k \mathrm{N}(\theta|\hat{\mu}_k, \hat{\sigma}_k^2)\right) p(y^*|x^*, \theta) d\theta \\ &\approx \frac{1}{T} \sum_{t=1}^{T} \sum_{k=1}^{K} \pi_k p(y^*|x^*, \theta_k^{(t)}) \\ &= q(y^*|x^*) \end{aligned} \tag{20}$$

Since classification tasks have a discrete nature, the predictive distribution is estimated by an average of discrete functions, which are frequently categorical probabilities.

$$\begin{aligned} p(y^*|x^*, x, y) &\approx \frac{1}{T} \sum_{t=1}^{T} \sum_{k=1}^{K} \pi_k p(y^*|x^*, \theta_k^{(t)}) \\ &= \frac{1}{T} \sum_{t=1}^{T} \sum_{k=1}^{K} \pi_k Cat(y^*|f(x^*, \theta_k^{(t)})) \\ &= \frac{1}{T} \sum_{t=1}^{T} \sum_{k=1}^{K} \pi_k \prod_{c=1}^{C} f^c(x^*, \theta_k^{(t)})^{y_c^*} \end{aligned} \tag{21}$$

With $\theta_k^{(t)} \sim \mathrm{N}(\theta|\hat{\mu}_k, \hat{\sigma}_k^2)$, for $t = 1, 2, ..., T$, and $k = 1, 2, ..., K$, $f(x^*, \theta_k^{(t)})$ is the output function of CNN, $f^c(x^*, \theta_k^{(t)}) = p(y_c^* = 1|f(x^*, \theta_k^{(t)}))$ with $\sum_{c=1}^{C} f^c(x^*, \theta_k^{(t)}) = 1$, and C is the number of classes in the output layer.

4.2 Uncertainty in CNN with GMM

Uncertainty estimation in deep neural networks is crucial for decision-making, especially in tasks that require a high degree of credibility and reliability. Generally, there are two types of uncertainty: aleatoric uncertainty and epistemic uncertainty. Aleatoric uncertainty is an irreducible quantity resulting from the noise generated by the data collection method. Regarding epistemic uncertainty, it results from model predictions when there is little observed data, but it can be reduced if more data are available. In general, Bayesian techniques provide an effective framework for estimating uncertainties by computing the variance of the predictive distribution over the variational posterior, as shown below:

$$\begin{aligned} \mathbf{V}_{\mathbf{q}_{\hat{\boldsymbol{\phi}}}(\boldsymbol{\theta})}[\mathbf{p}(\mathbf{y}^*|\mathbf{x}^*, \mathbf{x}, \mathbf{y})] &= \mathbf{E}_{\mathbf{q}_{\hat{\boldsymbol{\phi}}}(\boldsymbol{\theta})}[\mathbf{y}^* \mathbf{y}^{*\mathbf{T}}] - \mathbf{E}_{\mathbf{q}_{\hat{\boldsymbol{\phi}}}(\boldsymbol{\theta})}[\mathbf{y}^*] \mathbf{E}_{\mathbf{q}_{\boldsymbol{\phi}}(\boldsymbol{\theta})}[\mathbf{y}^*]^{\mathbf{T}} \\ &= \underbrace{\int_{\boldsymbol{\theta}} \left[\mathbf{diag}\left(\mathbf{E}_{\mathbf{p}(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\theta})}[\mathbf{y}^*]\right) - \mathbf{E}_{\mathbf{p}(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\theta})}[\mathbf{y}^*] \mathbf{E}_{\mathbf{p}(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\theta})}[\mathbf{y}^*]^{\mathbf{T}}\right] \mathbf{q}_{\hat{\boldsymbol{\phi}}}(\boldsymbol{\theta}) \mathbf{d}\boldsymbol{\theta}}_{\mathbf{Aleatoric-Uncertainty}} \\ &+ \underbrace{\int_{\boldsymbol{\theta}} \left(\mathbf{E}_{\mathbf{p}(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\theta})}[\mathbf{y}^*] - \mathbf{E}_{\mathbf{q}_{\hat{\boldsymbol{\phi}}}(\boldsymbol{\theta})}[\mathbf{y}^*]\right) \left(\mathbf{E}_{\mathbf{p}(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\theta})}[\mathbf{y}^*] - \mathbf{E}_{\mathbf{q}_{\hat{\boldsymbol{\phi}}}(\boldsymbol{\theta})}[\mathbf{y}^*]\right)^{\mathbf{T}} \mathbf{q}_{\hat{\boldsymbol{\phi}}}(\boldsymbol{\theta}) \mathbf{d}\boldsymbol{\theta}}_{\mathbf{Epistemic-Uncertainty}} \end{aligned} \tag{22}$$

For a review of the last formula's proof, see: [36].

As seen in the last formula, the variance of the predictive distribution is the sum of the aleatoric and epistemic uncertainty. Despite this, calculating these two uncertainties remains difficult due to the intractability of the integrals in the last formula. To solve this problem, we can combine the approach described earlier in Sect. 1 (Eq. 15) with the method proposed by Kwon in [36] to estimate the uncertainty using a GMM, as illustrated below:

$$\mathbf{V}_{q_{\hat{\phi}}(\theta)}[p(y^*|x^*,x,y)] \approx \underbrace{\frac{1}{T}\sum_{t=1}^{T}\sum_{k=1}^{K}\pi_k\left[diag(\hat{p}_k^{(t)}) - \hat{p}_k^{(t)}\hat{p}_k^{(t)^T}\right]}_{Aleatoric-Uncertainty} + \underbrace{\frac{1}{T}\sum_{t=1}^{T}\sum_{k=1}^{K}\pi_k(\hat{p}_k^{(t)} - \bar{p})(\hat{p}_k^{(t)} - \bar{p})^T}_{Epistemic-Uncertainty} \quad (23)$$

Where, $q_{\hat{\phi}}(\theta) = \sum_{k=1}^{K}\pi_k \mathrm{N}(\theta|\hat{\mu}_k, \hat{\sigma}_k^2)$,

$\bar{p} = \sum_{k=1}^{K}\pi_k \frac{1}{T}\sum_{t=1}^{T}\hat{p}_k^{(t)}$, $\hat{p}_k^{(t)} = f(x^*, \theta_k^{(t)})$ is the output function of CNN, and $\theta_k^{(t)} \sim \mathrm{N}(\theta|\hat{\mu}_k, \hat{\sigma}_k^2)$.

5 Experiments

In this section, we will apply the LeNet-5 network (as described in Appendix C, Table 9) to the MNIST and Fashion MNIST datasets. Additionally, we will use the CNN model defined in Appendix C (Table 10) for the CIFAR-10 and SVHN datasets (as specified in Appendix A, Table 8). To approximate the posterior distribution, we will employ the GMM as the variational distribution (BBGMM).

5.1 Experimental setup

Implementing the BCNN requires first defining the prior distribution of all CNN parameters. In this regard, we have adopted a fully factorized Gaussian distribution with a zero mean and a prior standard deviation $\boldsymbol{\sigma_0} > \mathbf{0}$ as a prior distribution of the parameters, as shown below:

$$p(\theta) = p(F, b_c, \omega, b_l) = \prod_{i=1}^{P}\mathrm{N}(\theta_i|0, \sigma_0^2) \quad (24)$$

$$\log p(\theta) = \log p(F, b_c, \omega, b_l) = \sum_{i=1}^{P}\log\left(\mathrm{N}(\theta_i|0, \sigma_0^2)\right) \quad (25)$$

Where P is the number of all CNN parameters.

After executing the CNN-feedforward on the dataset, we compute the log-likelihood as follows:

$$\begin{aligned}\log p(D|\theta) &= \log p(D|cnn(x,\theta)) \\ &= \sum_{n=1}^{N}\log p(y_n|cnn(x_n,\theta)) \\ &= \sum_{n=1}^{N}\log Cat(cnn^1(x_n,\theta), ..., cnn^C(x_n,\theta))\end{aligned} \quad (26)$$

Where C is the number of classes in the output layer (C = 10 for all datasets).

Computing $\log p(D|\theta)$ gets difficult when dealing with a huge dataset (N is large). To address this issue, the mini-batch optimization technique has demonstrated its efficiency in training time by randomly dividing the training data D into small partitions of equal size $D_1, ..., D_S$ and using them to train the model at each epoch.

$$\log p(D_s|\theta) = \sum_{n=1}^{M}\log p(y_{s_n}|x_{s_n},\theta)$$

Where $\log p(D|\theta) = \sum_{s=1}^{S}\log p(D_s|\theta)$, $D_s = \{(x_{s_n}, y_{s_n})\}_{n=1}^{M}$, S is the number of partitions, and M is the size of each partition.

Next, we define the variational distribution that approximates the posterior distribution of parameters. In our situation, we considered it as a mixture of two factorized Gaussian distributions, as shown below:

$$\begin{aligned}q_\phi(\theta) &= \pi\mathrm{N}(\theta|\mu_1,\sigma_1^2) + (1-\pi)\mathrm{N}(\theta|\mu_2,\sigma_2^2) \\ &= \pi\prod_{i=1}^{P}\mathrm{N}(\theta_i|\mu_{1_i},\sigma_{1_i}^2) + (1-\pi)\prod_{i=1}^{P}\mathrm{N}(\theta_i|\mu_{2_i},\sigma_{2_i}^2)\end{aligned} \quad (27)$$

$$\log q_\phi(\theta) = \log\left(\pi\prod_{i=1}^{P}\mathrm{N}(\theta_i|\mu_{1_i},\sigma_{1_i}^2) + (1-\pi)\prod_{i=1}^{P}\mathrm{N}(\theta_i|\mu_{2_i},\sigma_{2_i}^2)\right) \quad (28)$$

Where $0 < \pi < 1$ is the mixture weight considered as a hyperparameter.

Finally, we can approximate the cost function of this model as follows:

$$\begin{aligned}&\mathscr{L}(D,\phi) \approx \hat{\mathscr{L}}_{MB}(\theta,\phi) \\ &= \frac{1}{T}\sum_{t=1}^{T}\Bigg[\pi\Big(\log q_\phi(\theta_1^{(t)}) - \log p(\theta_1^{(t)}) - \frac{N}{M}\sum_{n=1}^{M}\log p(y_{n_s}|x_{s_n},\theta_1^{(t)})\Big) \\ &+ (1-\pi)\Big(\log q_\phi(\theta_2^{(t)}) - \log p(\theta_2^{(t)}) - \frac{N}{M}\sum_{n=1}^{M}\log p(y_{s_n}|x_{s_n},\theta_2^{(t)})\Big)\Bigg]\end{aligned} \quad (29)$$

Where $\theta_1^{(t)} = \mu_1 + \sigma_1 \odot \epsilon_1^{(t)}$, and $\theta_2^{(t)} = \mu_2 + \sigma_2 \odot \epsilon_2^{(t)}$, where $\epsilon_1^{(t)} \sim \mathrm{N}(\epsilon_1|0, I)$, and $\epsilon_2^{(t)} \sim \mathrm{N}(\epsilon_2|0, I)$, for t = 1,...,T.

5.2 Results and analysis

This section provides an assessment of the performance of BBGMM method described in Algorithm 4 compared to existing methods (Frequentist approach, BBGaussian [58], and MC Dropout [18]) in classification tasks using MNIST, Fashion MNIST, CIFAR-10, and SVHN datasets (Appendix A, Table 6). We then evaluate the uncertainties associated with our proposed method for these datasets.

5.2.1 Datasets (Appendix A, Table 6)

We evaluate our method using the following datasets: **1. MNIST:** This well-established benchmark dataset comprises grayscale images representing handwritten digits. It contains 70,000 samples, each image sized at $1 \times 28 \times 28$ pixels. The dataset is strategically partitioned into training (50,000 samples), validation (10,000 samples), and test (10,000 samples) sets, facilitating essential aspects of model training, hyperparameter tuning, and performance evaluation.

2. Fashion MNIST: An alternative to MNIST, this dataset shifts focus to fashion products. Sharing structural similarities with MNIST, it encompasses 70,000 grayscale images of fashion items, each sized at $1 \times 28 \times 28$ pixels. Like MNIST, it is partitioned into training, validation, and test sets, comprising 50,000, 10,000, and 10,000 samples, respectively.

3. CIFAR-10: Representing a more intricate challenge than MNIST and Fashion MNIST, CIFAR-10 consists of 60,000 color images, each with dimensions of $3 \times 32 \times 32$. The dataset covers ten distinct object categories and is is split into training (40,000 samples), validation (10,000 samples), and test (10,000 samples) sets to support effective model development and evaluation.

4. SVHN: Focused on digit recognition within real-world images, the SVHN dataset captures house numbers from street views. Comprising 73,257 color images sized at $3 \times 32 \times 32$ pixels. This dataset includes a range of digits from 0 to 9. The dataset is partitioned into training (53,257 samples), validation (20,000 samples), and test (26,032 samples) sets, enabling a comprehensive assessment of model performance.

5.2.2 Results on MNIST and fashion MNIST

Table 2 compares the training, validation, and test accuracies (in percent) of LeNet-5 on the MNIST and Fashion MNIST datasets, evaluating our method against frequentist, MC dropout, and BBGaussian models.

Overall, the table shows comparable results between the models. On MNIST, the frequentist model obtained the highest training accuracy of 99.97% compared with the other models, with a test accuracy of 98.54%.

On the other hand, our model achieved a training accuracy of 99.87% and a test accuracy of 98.85%. For Fashion MNIST, our model outperformed the others with a test accuracy of 89.02%, indicating that the BBGMM model is more accurate and reliable on Fashion MNIST test data.

Figs. 3 and 4 display the evolution of validation accuracy during LeNet-5 training on the MNIST and Fashion MNIST datasets, comparing previous approaches with our model. Notably, the validation accuracy curves for all models show comparable performance, but with a preference for the BBGMM model (blue lines).

Table 3 compares the training and validation errors obtained by training the LeNet-5 network on the MNIST and Fashion MNIST datasets using the BBGaussian and BBGMM models. The results indicate that our proposed model achieves lower training and validation errors than the BBGaussian model for both datasets. This suggests that incorporating a mixture model (GMM) within the Bayes by

Table 2 Comparison of accuracies between Frequentist, MC dropout, BB-Gaussian, and BBGMM models on the MNIST, and Fashion MNIST datasets

Datasets	Models	Accuracies		
		Training accuracy %	Validation accuracy %	Test accuracy %
MNIST	Frequentist LeNet-5	**99.97**	98.68	98.54
	MC dropout LeNet-5	99.13	98.46	98.50
	BB Gaussian LeNet-5	99.83	98.73	98.67
	BBGMM LeNet-5	99.87	98.87	**98.85**
Fashion-MNIST	Frequentist LeNet-5	**90.91**	89.82	88.66
	MC dropout LeNet-5	90.20	89.88	88.65
	BB Gaussian LeNet-5	90.11	89.96	88.58
	BBGMM LeNet-5	90.77	90.33	**89.02**

Bold shows the results that is more significant

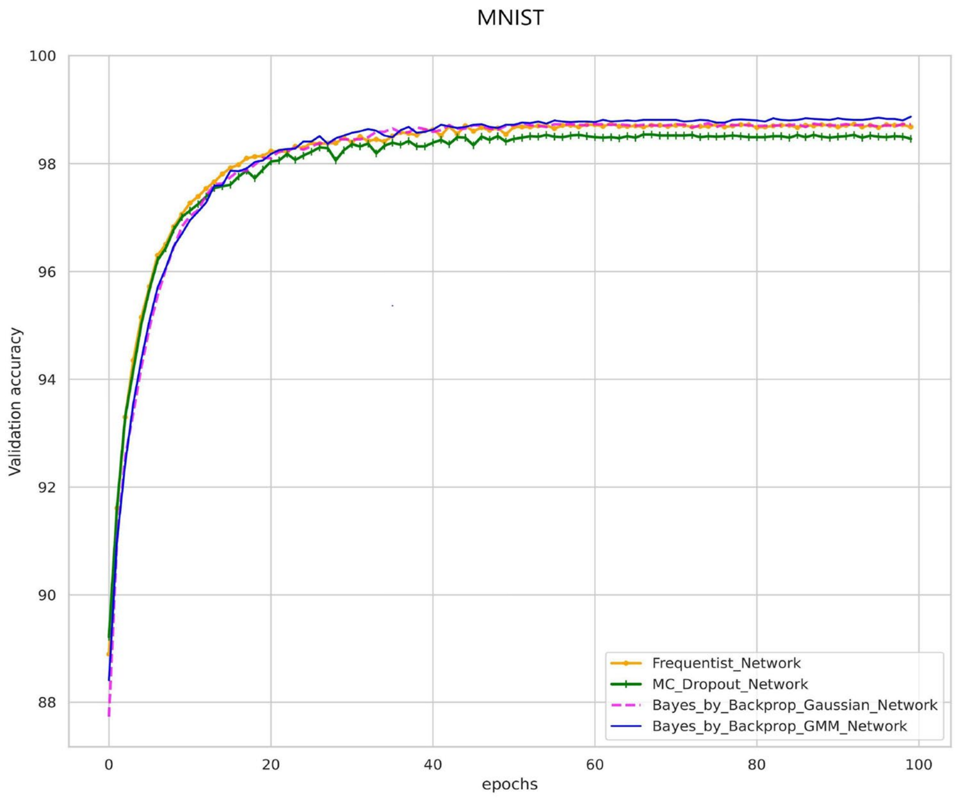


Fig. 3 Comparison of validation accuracies on MNIST

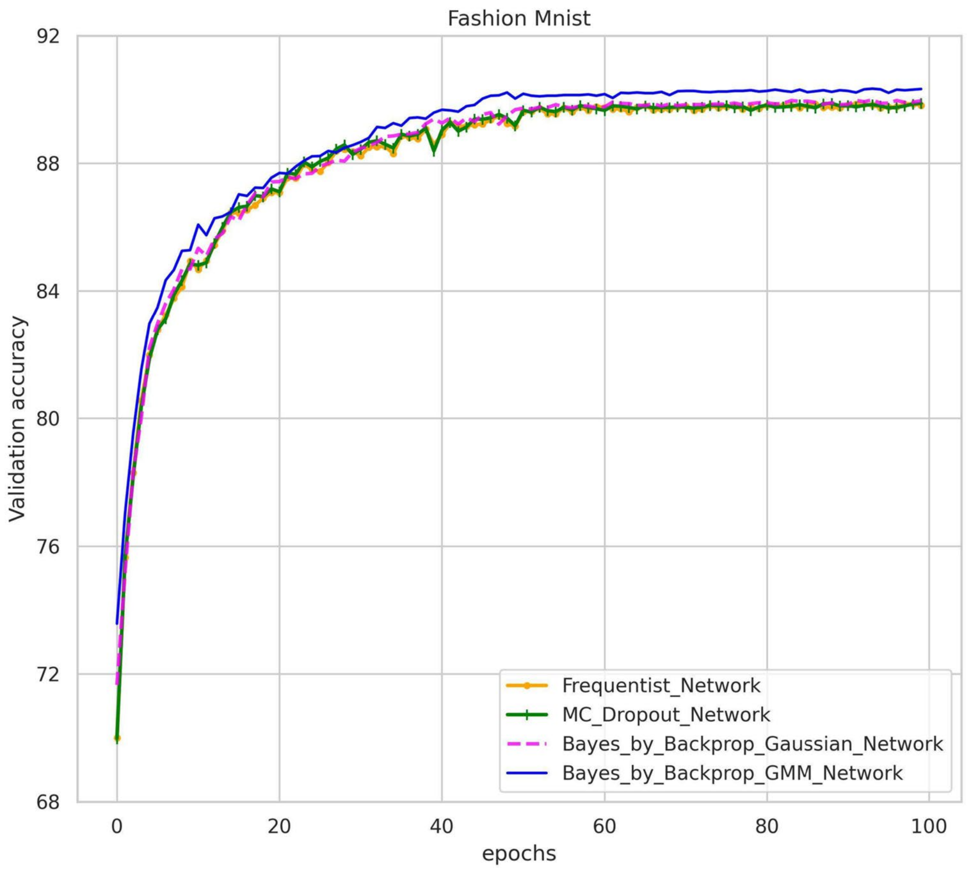


Fig. 4 Comparison of validation accuracies Fashion MNIST

Table 3 Comparison of errors on the MNIST, and Fashion MNIST datasets

Datasets	Models	Accuracies	
		Training error %	Validation error
MNIST	BB Gaussian	2.53034	2.59654
	BBGMM	1.06146	**1.11173**
Fashion MNIST	BB Gaussian	1.92743	1.94768
	BBGMM	0.89286	**0.90135**

Bold shows the results that is more significant

Table 4 Comparison of accuracies between Frequentist, MC dropout, BB-Gaussian, and BBGMM models on CIFAR-10, and SVHN datasets

Datasets	Models	Accuracies		
		Training accuracy %	Validation accuracy %	Test accuracy %
CIFAR-10	Frequentist CNN	**84.20**	78.82	79.52
	MC dropout CNN	81.08	79.46	79.52
	BB Gaussian CNN	81.49	77.23	78.68
	BBGMM CNN	83.76	79.86	**80.60**
SVHN	Frequentist CNN	93.88	92.12	93.21
	MC dropout CNN	92.42	92.20	93.16
	BB Gaussian CNN	93.71	92.53	93.36
	BBGMM CNN	**94.70**	93.36	**94.53**

Backprop method for training CNN on the MNIST and Fashion MNIST datasets yields improved performance than using a single mode distribution, such as the Gaussian distribution.

5.2.3 Results on CIFAR-10 and SVHN

In this section, we have used a CNN network consisting of three blocks similar to the VGG blocks and two fully connected layers. Each block consists of two convolutional layers followed by max-pooling, and we adopted ReLu as an activation function (see Appendix C (Table 10)).

Table 4 presents a comparison of accuracies between the previous models (Frequentist, MC dropout, BB-Gaussian) and the BBGMM method, using the CNN model described in Appendix C (Table 10), on the CIFAR-10 and SVHN datasets. Table 4 shows that our method performed better than other models. For the CIFAR-10 dataset, after 100 epochs, our BBGMM model achieved the highest test accuracy of 80.60%, with a corresponding training accuracy of 83.76%. On the other hand, the MC dropout model achieved a test accuracy of 79.52% and a training accuracy of 81.08%. As for the BBGaussian method, it yielded a lower test accuracy of 78.68%. For the SVHN dataset, our BBGMM model achieved the highest test accuracy of 94.53%, outperforming the frequentist, MC dropout, and BBGaussian models, which achieved test accuracies of 93.21%, 93.16%, and 93.36%, respectively.

Figs. 5 and 6 illustrate the evolution of validation accuracies during CNN training on the CIFAR-10 and SVHN datasets, respectively. Generally, the results show comparable performance on the two datasets, with a slight improvement observed for our BGMM method, as indicated by the blue lines in the figures.

Fig. 7 shows the progression of validation error during CNN training on the CIFAR-10 and SVHN datasets using two different variational distributions: the Bayes by Backprop method with a single Gaussian distribution (BBGaussian, orange lines) and the same method with a mixture model of two Gaussian distributions (BBGMM, bleu lines). The figure indicates that both methods converge as training progresses on both datasets. However, it is worth noting that the BBGMM model consistently achieves a lower validation error than the BBGaussian model. This lower validation error indicates that the BBGMM method performs more accurately on the validation set than the BBGaussian method.

Fig. 8 displays the evolution of the probability density of a weight taken from the last layer of the CNN over training iterations. The weight is sampled from the two Gaussian distributions employed in the mixture for classifying the CIFAR-10 images. Figure 9 shows that during the first ten training epochs, the weight density appears unimodal due to the proximity of the means of the two Gaussian distributions used.

However, as the training progresses, the density is transformed into a bimodal distribution, with each mode corresponding to a different mean. This observation highlights the power of approximating the posterior distribution for this case using a mixture of Gaussian distributions rather than adopting an unimodal distribution such as the Gaussian distribution.

Figures 9, and 10 depict the convergence of standard deviations $\boldsymbol{\sigma_1}$ and $\boldsymbol{\sigma_2}$ for the Gaussian mixture distribution represented in Fig. 8. It is noticed in Fig. 9 that the standard deviations of each component of the binary mixture of Gaussian distributions decrease over epochs, indicating that the uncertainty of the model reduces as more new data is received. Thus, the model gets more reliable with its training.

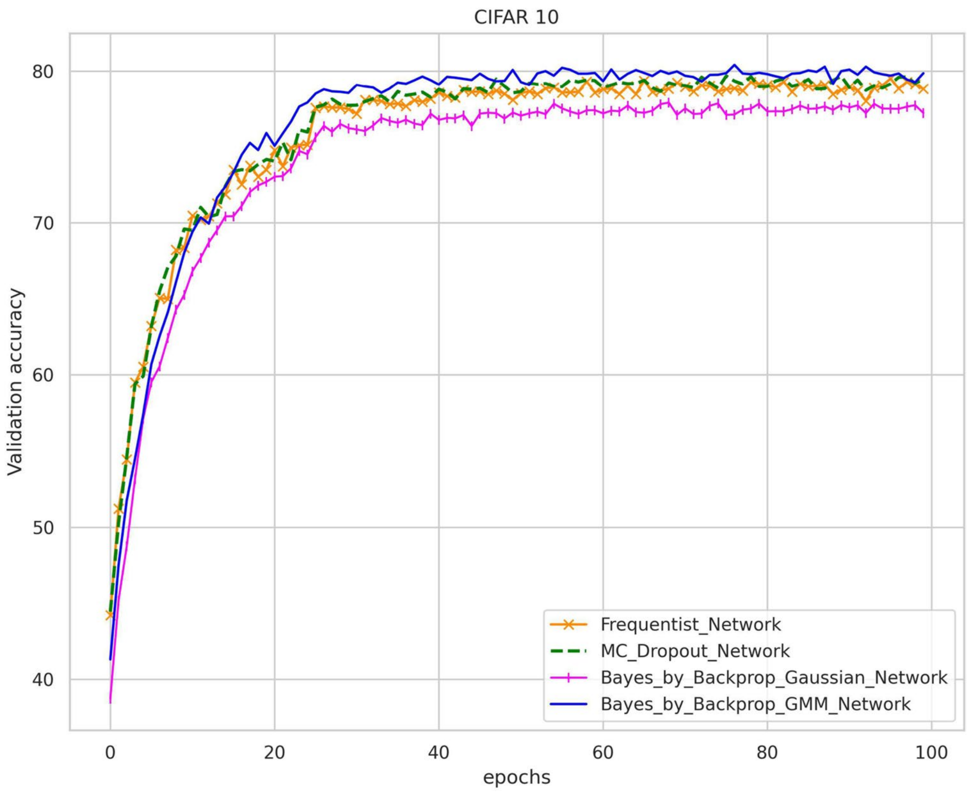


Fig. 5 Comparison of validation accuracies on CIFAR 10

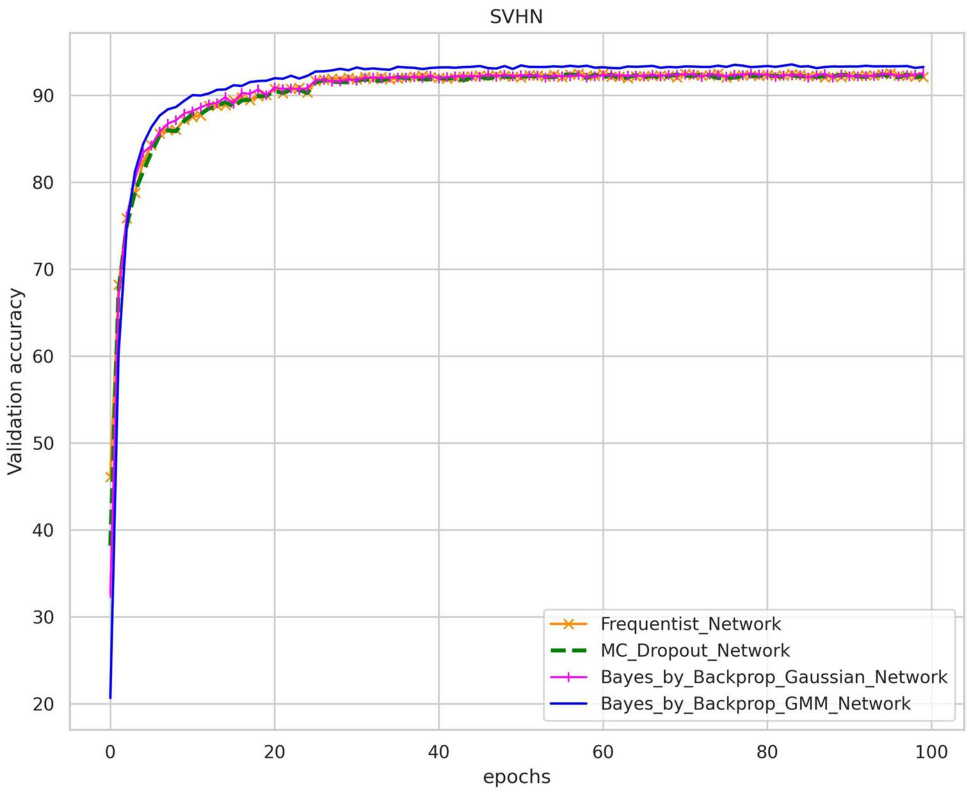


Fig. 6 Comparison of validation accuracies on SVHN

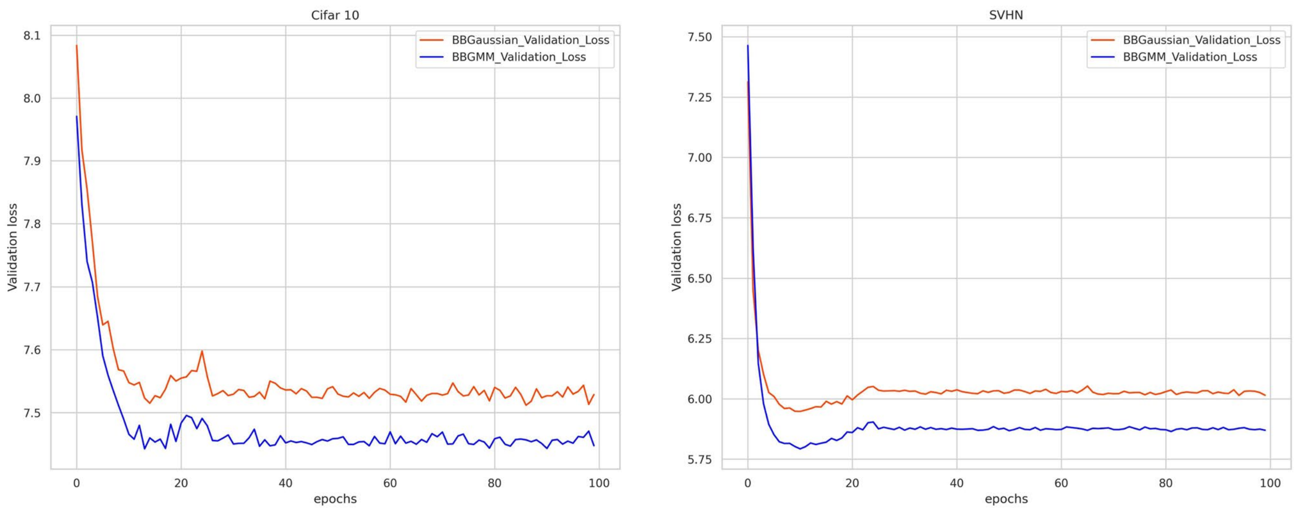
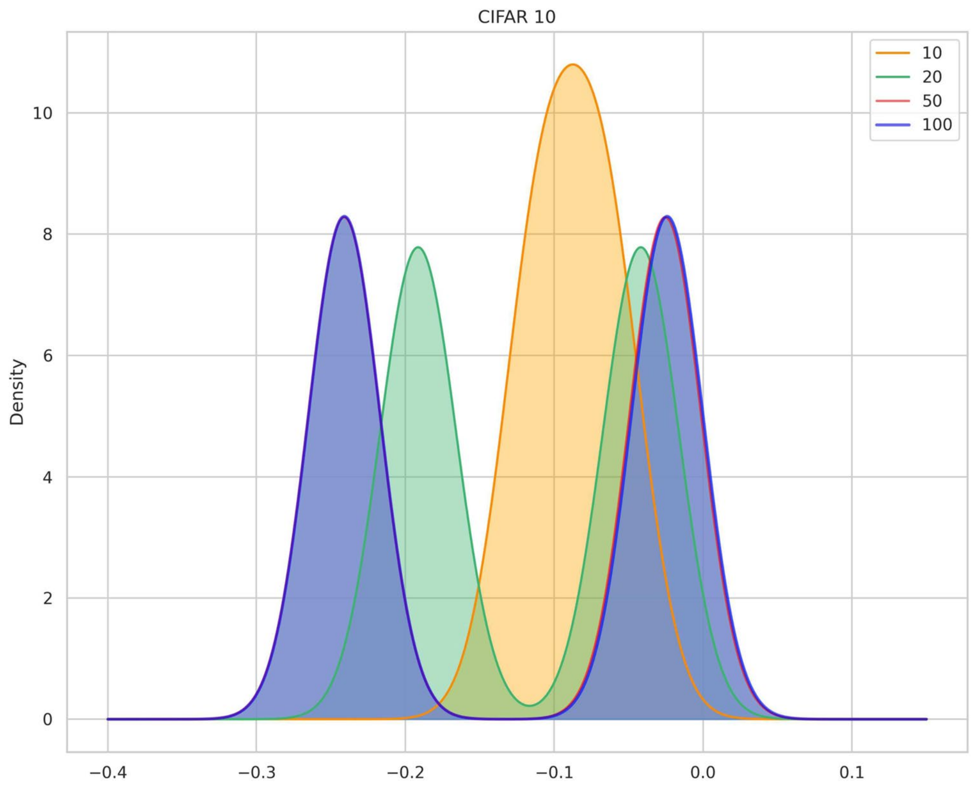


Fig. 7 Comparison of validation losses on CIFAR-10, and SVHN

Fig. 8 Probability density evolution



5.2.4 Uncertainty estimation

Figure 11 shows the BBGMM model prediction for two randomly selected images in the test set from the MNIST and Fashion-MNIST datasets and estimates the BBGMM model uncertainties using Eq. 23 for these two images. It illustrates that the BBGMM model can predict the classification of the picture extracted from MNIST, but with a percentage of aleatoric uncertainty owing to the noise in this image. However, the BBGMM model failed to correctly classify the picture taken from Fashion-MNIST since it differed from the MNIST dataset used during training, which explains the significant value of epistemic uncertainty in this image. Also, it shows a significant value of aleatoric uncertainty in the last image due to the noise present in this image.

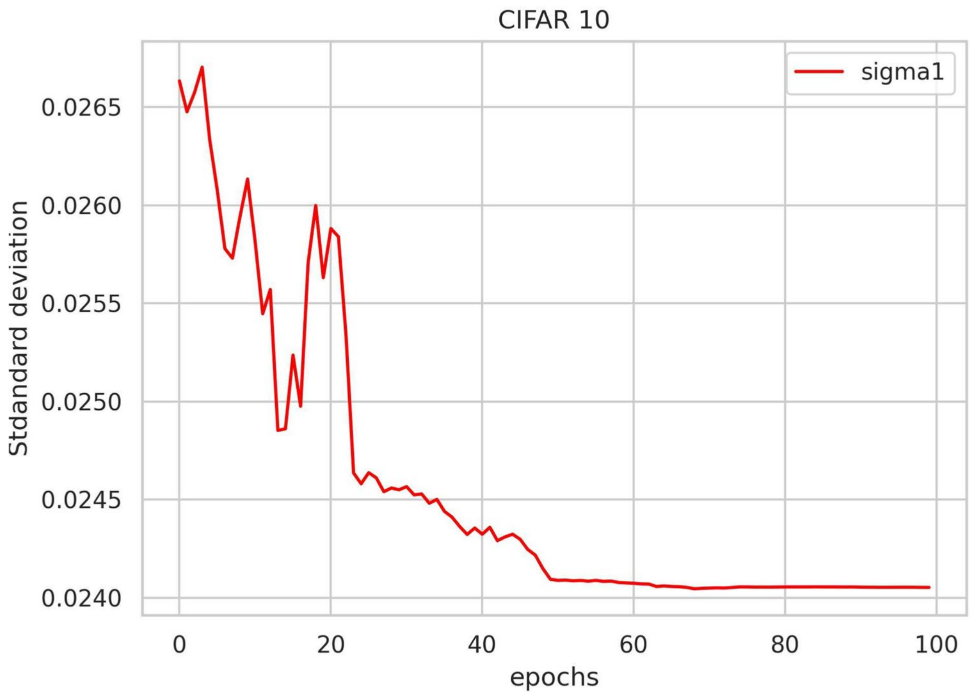


Fig. 9 Standard deviation (σ_1) evolution

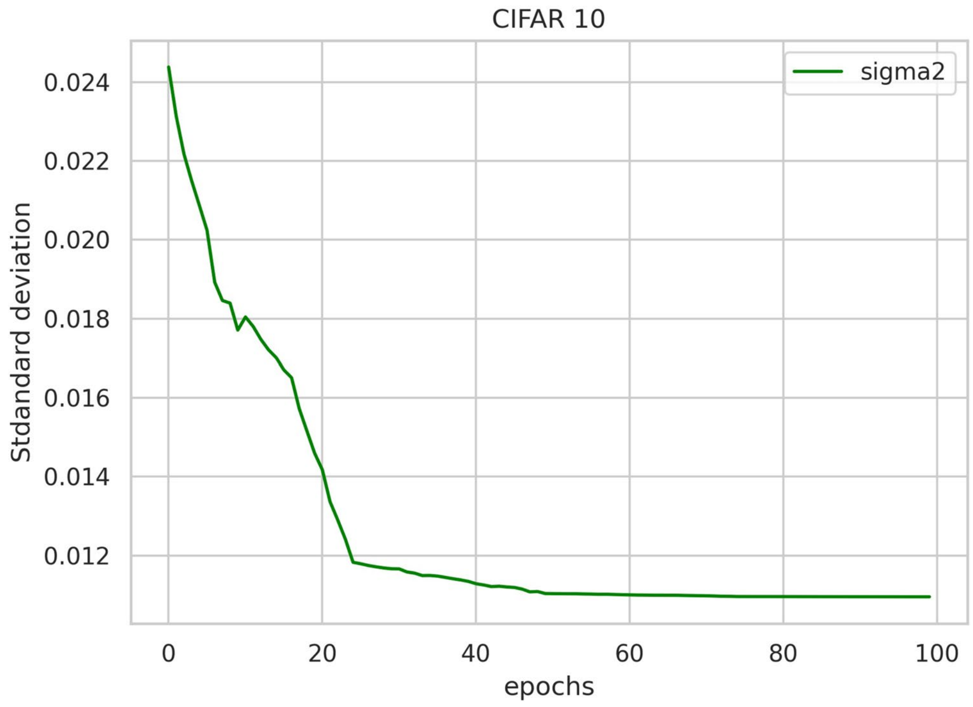


Fig. 10 Standard deviation (σ_2) evolution

Table 5 compares the average epistemic and aleatoric uncertainties for the BBGMM model applied to the MNIST, Fashion MNIST, CIFAR-10 and SVHN datasets using the LeNet-5 network. The results indicate that the aleatoric uncertainty of CIFAR-10 is over thirty times higher than that of MNIST, while the epistemic uncertainty of CIFAR-10 is more than thirteen times greater than that of MNIST. The variations in aleatoric and epistemic uncertainty between all datasets can be attributed to the performance of the LeNet5-BBGMM model on each specific dataset. In other words, the model's accuracy

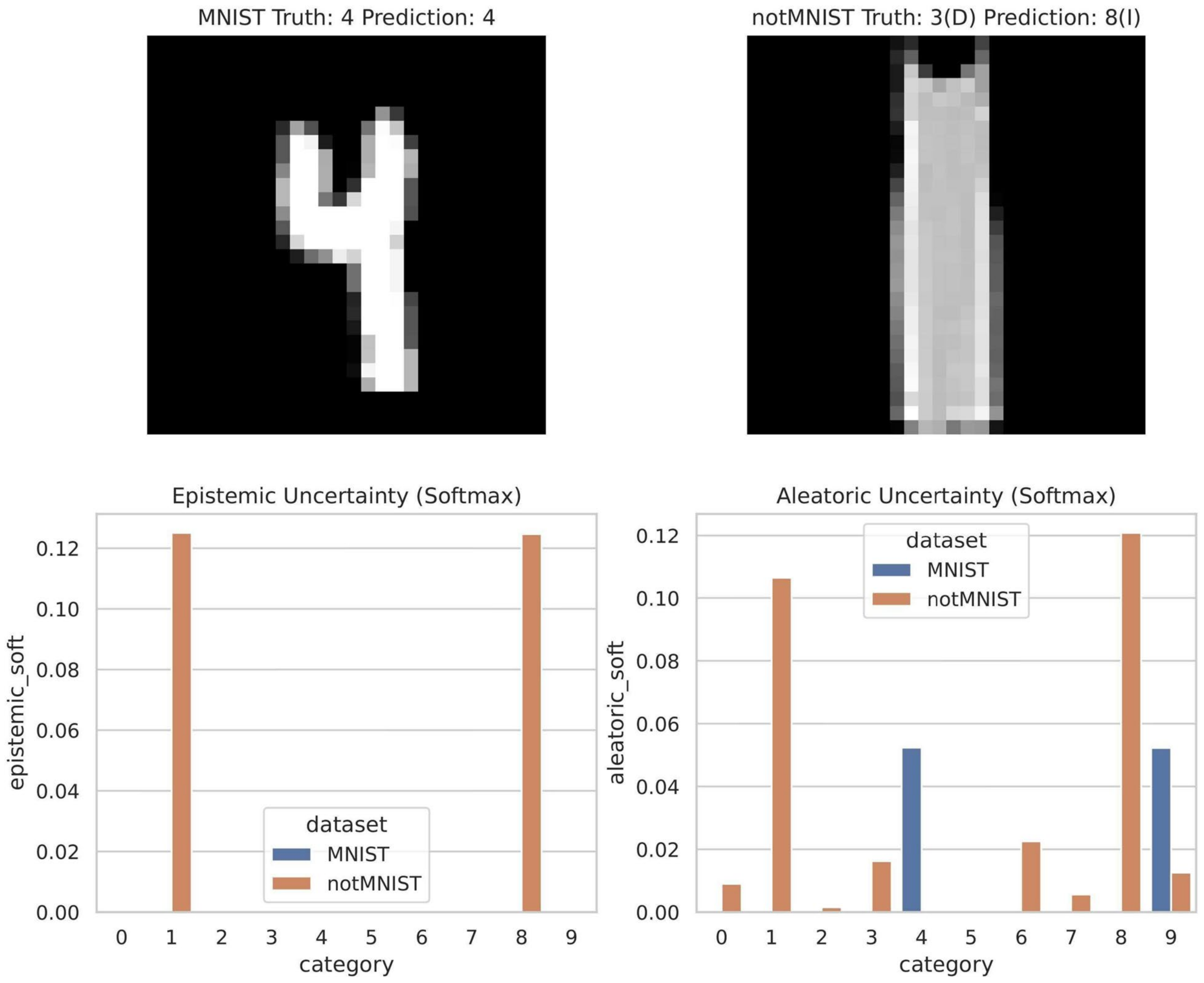


Fig. 11 The image on the top left is an image of number 4 taken from the test set for MNIST. The image on the top right is an image of a dress (class 3) taken from the test set for Fashion-MNIST. The bottom left image shows the percentage of epistemic uncertainty for the BBGMM model predictions of the previous two pictures, where a high value of epistemic uncertainty appears between classes 1 and 8 for the image taken from Fashion-Mnist, as it is completely different from the MNIST set, but it has a high similarity between it and both numbers 1 and 8, so the BBGMM model predicts uncertain outputs for this image between these two classes. The picture at the bottom right presents the proportion of aleatoric uncertainty for the BBGMM model predictions in the previous two images, where varying values of this uncertainty appear between the two images resulting from the noise in these two images. For example, we can explain the appearance of aleatoric uncertainty between the numbers 4 and 9 for the image taken from MNIST by the presence of a significant similarity between these numbers in this image

Table 5 Uncertainty estimation

Datasets	Aleatoric uncertainty	Epistemic uncertainty
MNIST	0.001229	0.000307
Fashion MNIST	0.01394	0.0004003
CIFAR-10	0.037939	0.004080
SVHN	0.012833	0.037308

on the test data directly affects its reliability, with lower uncertainty values indicating higher reliability.

5.3 Training time complexity

The training time of the BGMM for CNNs can be estimated as the training time required for a Bayesian CNN with a single Gaussian distribution having a similar structure, multiplied by K, where K represents the number of components in the mixture. It is important to note that this approach can be computationally expensive, especially for high-dimensional models like CNNs. However, the advantage of the BGMM lies in its ability to reparameterize the Gaussian mixture, providing differentiable estimates. This, in turn, leads to a reduction in the variance of the gradients for our objective function, resulting in more accurate and reliable predictions compared to other models.

6 Conclusion and discussion

The BBGMM model can be viewed as an extension of the Bayes by Backprop method for mixture models (Gaussian mixture model) to approximate the posterior distribution for convolutional neural networks. We followed how to reparameterize the mixture model if all of its components are reparametrizable, which is available in Gaussian mixture models. We then tried to estimate the aleatoric and epistemic uncertainties of mixture models for classification tasks.

Next, the experimental results showed that the BBGMM model performed well compared to the other methods, as it achieved significant test accuracy over all datasets.

We also saw that the BBGMM model increased the credibility of convolutional neural network outcomes by quantifying uncertainty in network weights from the mixture model.

Although the BBGMM method has produced positive results, it is crucial to note that the extent of these results is currently insufficient. This implies that while the BBGMM method is promising, there is still considerable potential to improve it to achieve more substantial and robust results. The second problem is that the BBGMM method doubles the number of parameters that vary as a function of the number of components in the mixture, which increases the complexity of the model and makes it inflexible or difficult to apply in practical applications, particularly when the number of mixture components increases.

This motivates us to improve the BBGMM model and discover complementary solutions to reduce the model's complexity.

Appendix A: Datasets

See Table 6.

Table 6 Datasets informations

Informations	Datasets			
	MNIST [39]	Fashion MNIST [68]	CIFAR-10 [33]	SVHN [50]
Content	Handwritten digits images	Fashion products	Objects pictures	Digits images
Number	70000	70000	60000	73257
Type	Grayscale images	Grayscale images	Colour images	Colour images
Size	$1 \times 28 \times 28$	$1 \times 28 \times 28$	$3 \times 32 \times 32$	$3 \times 32 \times 32$
Class	10	10	10	10
Train set	50000	50000	40000	53257
Validation set	10000	10000	10000	20000
Test set	10000	10000	10000	26032

Appendix B: Hyperparameters

See Tables 7, 8.

Table 7 Hyperparameters using for MNIST and Fashion MNIST

Hyperameters	Value
Epochs	100
Batch_size	128
Sample train	2
Sample test	10
Loss	Cross-Entropy
Optimizer	Adam
Learning rate	0.00015
Prior std σ_0	0.1
Mixture weight π	0.5

Table 8 Hyperparameters using for CIFAR-10 and SVHN

Hyperameters	Value
Epochs	100
Batch_size	128
Sample train	2
Sample test	10
Loss	Cross-Entropy
Optimizer	Adam
Learning rate	0.001
Prior std σ_0	0.1
Mixture weight π	0.5

Appendix C: CNN architectures

See Tables 9, 10.

Table 9 LeNet-5

Layers	Layer type	Layer size	Stride	Padding	Activation function
1	Convolution	$5 \times 5 \times 6$	1	2	Tanh
2	Max pooling	2×2	2	0	
3	Convolution	$5 \times 5 \times 16$	1	0	Tanh
4	Max pooling	2×2	2	0	
5	Fully connected	400×120			Tanh
6	Fully connected	120×84			Tanh
7	Fully connected	84×10			Softmax

Table 10 CNN architecture

Layers	Layer type	Layer size	Stride	Padding	Activation function
1	Convolution	$3 \times 3 \times 16$	1	Same	ReLu
2	Convolution	$3 \times 3 \times 16$	1	Same	ReLu
3	Max pooling	2×2	2	0	
4	Convolution	$3 \times 3 \times 32$	1	Same	ReLu
5	Convolution	$3 \times 3 \times 32$	1	Same	ReLu
6	Max pooling	2×2	2	0	
7	Convolution	$3 \times 3 \times 64$	1	Same	ReLu
8	Convolution	$3 \times 3 \times 64$	1	Same	ReLu
9	Max pooling	2×2	2	0	
10	Fully connected	1024×64			ReLu
11	Fully connected	64×10			Softmax

Data availibility Datasets analyzed during the current study are available in the UCI machine learning repository.

Declarations

Conflict of interest The authors declare that they have no conflicts of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Abdar M, Pourpanah F, Hussain S, Rezazadegan D, Liu L, Ghavamzadeh M, Fieguth P, Cao X, Khosravi A, Acharya UR et al (2021) A review of uncertainty quantification in deep learning: techniques, applications and challenges. Inf Fusion 76:243–297
2. Albawi S, Mohammed TA, Al-Zawi S (2017) Understanding of a convolutional neural network. In: 2017 international conference on engineering and technology (ICET). IEEE, pp 1–6
3. Alzubaidi L, Zhang J, Humaidi AJ, Al-Dujaili A, Duan Y, Al-Shamma O, Santamaría J, Fadhel MA, Al-Amidie M, Farhan L (2021) Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data 8:1–74
4. Barber D, Bishop CM (1998) Ensemble learning in Bayesian neural networks. Nato ASI Ser F Comput Syst Sci 168:215–238
5. Bardenet R, Doucet A, Holmes C (2017) On Markov chain monte Carlo methods for tall data. J Mach Learn Res 18(1):1515–1557
6. Bishop CM et al (1995) Neural networks for pattern recognition. Oxford University Press, Oxford
7. Blei DM, Kucukelbir A, McAuliffe JD (2017) Variational inference: a review for statisticians. J Am Stat Assoc 112(518):859–877
8. Blundell C, Cornebise J, Kavukcuoglu K, Wierstra D (2015) Weight uncertainty in neural network. In: International conference on machine learning. PMLR, pp 1613–1622
9. Boué L (2018) Deep learning for pedestrians: backpropagation in CNNS. arXiv preprint arXiv:1811.11987
10. Chan A, Alaa A, Qian Z, Van Der Schaar M (2020) Unlabelled data improves Bayesian uncertainty calibration under covariate shift. In: International conference on machine learning. PMLR, pp 1392–1402
11. Chandra R, Chen R, Simmons J (2023) Bayesian neural networks via MCMC: a python-based tutorial. arXiv preprint arXiv:2304.02595
12. Chib S, Greenberg E (1995) Understanding the metropolis-hastings algorithm. Am Stat 49(4):327–335
13. Christopher M (2006) Pattern recognition and machine learning. Springer, New York
14. Depeweg S, Hernandez-Lobato J-M, Doshi-Velez F, Udluft S (2018) Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning. In: International conference on machine learning. PMLR, pp 1184–1193
15. Figurnov M, Mohamed S, Mnih A (2018) Implicit reparameterization gradients. In: Advances in neural information processing systems, vol 31
16. Gal Y. Uncertainty in deep learning
17. Gal Y, Ghahramani Z (2015) Bayesian convolutional neural networks with Bernoulli approximate variational inference. arXiv preprint arXiv:1506.02158
18. Gal Y, Ghahramani Z (2016) Dropout as a Bayesian approximation: representing model uncertainty in deep learning. In: International conference on machine learning. PMLR, pp 1050–1059
19. Goan E, Fookes C (2020) Bayesian neural networks: an introduction and survey. In: Case studies in applied Bayesian data science. Springer, pp 45–87
20. Graves A (2011) Practical variational inference for neural networks. In: Advances in neural information processing systems, vol 24
21. Graves A (2016) Stochastic backpropagation through mixture density distributions. arXiv preprint arXiv:1607.05690

22. Greenspan H, Van Ginneken B, Summers RM (2016) Guest editorial deep learning in medical imaging: overview and future promise of an exciting new technique. IEEE Trans Med Imaging 35(5):1153–1159
23. Guo W, Mu D, Xu J, Su P, Wang G, Xing X (2018) Lemna: explaining deep learning based security applications. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, pp 364–379
24. Hastings WK (1970) Monte Carlo sampling methods using Markov chains and their applications
25. Hernández-Lobato JM, Adams R (2015) Probabilistic backpropagation for scalable learning of Bayesian neural networks. In: International conference on machine learning. PMLR, pp 1861–1869
26. Hinton G, van Camp D (1993) Keeping neural networks simple by minimising the description length of weights. In: Proceedings of COLT-93, pp 5–13
27. Jospin LV, Laga H, Boussaid F, Buntine W, Bennamoun M (2022) Hands-on Bayesian neural networks-a tutorial for deep learning users. IEEE Comput Intell Mag 17(2):29–48
28. Ker J, Wang L, Rao J, Lim T (2017) Deep learning applications in medical image analysis. IEEE Access 6:9375–9389
29. Khairnar P, Thiagarajan P, Ghosh S (2020) A modified Bayesian convolutional neural network for breast histopathology image classification and uncertainty quantification
30. Kingma DP, Welling M (2019) An introduction to variational autoencoders. arXiv preprint arXiv:1906.02691
31. Kingma DP, Salimans T, Welling M (2015) Variational dropout and the local reparameterization trick. Adv Neural Inf Process Syst 28:2575–2583
32. Kristiadi A, Hein M, Hennig P (2020) Being Bayesian, even just a bit, fixes overconfidence in Relu networks. In: International conference on machine learning, pp 5436–5446. PMLR
33. Krizhevsky A, Hinton G, et al (2009) Learning multiple layers of features from tiny images
34. Krzywinski M, Altman N (2013) Importance of being uncertain. Nat Methods 10(9):809–811
35. Kullback S, Leibler RA (1951) On information and sufficiency. Ann Math Stat 22(1):79–86
36. Kwon Y, Won J-H, Kim BJ, Paik MC (2018) Uncertainty quantification using Bayesian neural networks in classification: application to ischemic stroke lesion segmentation
37. Lampinen J, Vehtari A (2001) Bayesian approach for neural networks-review and case studies. Neural Netw 14(3):257–274
38. LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD (1989) Backpropagation applied to handwritten zip code recognition. Neural Comput 1(4):541–551
39. LeCun Y, Cortes C, Burges C (2010) Mnist handwritten digit database
40. Lee W, Yu H, Yang H (2018) Reparameterization gradient for non-differentiable models. In: Advances in neural information processing systems, vol 31
41. Li Z, Liu F, Yang W, Peng S, Zhou J (2021) A survey of convolutional neural networks: analysis, applications, and prospects. IEEE Trans Neural Netw Learn Syst
42. MacKay DJ (1992) A practical Bayesian framework for backpropagation networks. Neural Comput 4(3):448–472
43. Meena G, Mohbey KK, Kumar S, Lokesh K (2023) A hybrid deep learning approach for detecting sentiment polarities and knowledge graph representation on monkeypox tweets. Decision Anal J 7:100243
44. Minka TP (2013) Expectation propagation for approximate Bayesian inference. arXiv preprint arXiv:1301.2294
45. Mitros J, Mac Namee B (2019) On the validity of Bayesian neural networks for uncertainty estimation. arXiv preprint arXiv:1912.01530
46. Mohbey KK, Meena G, Kumar S, Lokesh K (2023) A CNN-LSTM-based hybrid deep learning approach for sentiment analysis on Monkeypox tweets. New Gener Comput 1–19
47. Morningstar W, Vikram S, Ham C, Gallagher A, Dillon J (2021) Automatic differentiation variational inference with mixtures. In: International conference on artificial intelligence and statistics. PMLR, pp 3250–3258
48. Mostafa B, Hassan R, Mohammed H, Tawfik M (2023) A review of variational inference for Bayesian neural network. In: International conference on artificial intelligence & industrial applications, pp 231–243. Springer
49. Neal RM et al (2011) MCMC using Hamiltonian dynamics. Handbook of Markov chain monte Carlo 2(11):2
50. Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng AY (2011) Reading digits in natural images with unsupervised feature learning
51. O'Shea K, Nash R (2015) An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458
52. Qi CR, Su H, Mo K, Guibas LJ (2017) Pointnet: deep learning on point sets for 3D classification and segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 652–660
53. Reynolds DA (2009) Gaussian mixture models. Encyclopedia Biomet 741:659–663
54. Ritter H, Botev A, Barber D (2018) A scalable Laplace approximation for neural networks. In: 6th international conference on learning representations, ICLR 2018-conference track proceedings, vol 6
55. Roeder G, Wu Y, Duvenaud DK (2017) Sticking the landing: simple, lower-variance gradient estimators for variational inference. In: Advances in neural information processing systems, vol 30
56. Roy A, Sun J, Mahoney R, Alonzi L, Adams S, Beling P (2018) Deep learning detecting fraud in credit card transactions. In: 2018 Systems and information engineering design symposium (SIEDS). IEEE, pp 129–134
57. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323(6088):533–536
58. Shridhar K, Laumann F, Liwicki M (2019) A comprehensive guide to Bayesian convolutional neural network with variational inference. arXiv preprint arXiv:1901.02731
59. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556
60. Sun S (2013) A review of deterministic approximate inference techniques for Bayesian machine learning. Neural Comput Appl 23(7):2039–2050
61. Sun S, He S (2019) Generalizing expectation propagation with mixtures of exponential family distributions and an application to Bayesian logistic regression. Neurocomputing 337:180–190
62. Titterington D (2004) Bayesian methods for neural networks and related models. Stat Sci 128–139
63. Tobore I, Li J, Yuhang L, Al-Handarish Y, Kandwal A, Nie Z, Wang L et al (2019) Deep learning intervention for health care challenges: some biomedical domain considerations. JMIR Mhealth Uhealth 7(8):e11966
64. Wang J, Ma Y, Zhang L, Gao RX, Wu D (2018) Deep learning for smart manufacturing: methods and applications. J Manuf Syst 48:144–156
65. Wang Y-H, Su W-H (2022) Convolutional neural networks in computer vision for grain crop phenotyping: a review. Agronomy 12(11):2659
66. Wilson AG, Izmailov P (2020) Bayesian deep learning and a probabilistic perspective of generalization. arXiv preprint arXiv:2002.08791
67. Winn J, Bishop CM, Jaakkola T (2005) Variational message passing. J Mach Learn Res 6(4)

68. Xiao H, Rasul K, Vollgraf R (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747
69. Zhang C, Bütepage J, Kjellström H, Mandt S (2018) Advances in variational inference. IEEE Trans Pattern Anal Mach Intell 41(8):2008–2026
70. Zhao J, Liu X, He S, Sun S (2020) Probabilistic inference of Bayesian neural networks with generalized expectation propagation. Neurocomputing 412:392–398
71. Zhou X, Liu H, Pourpanah F, Zeng T, Wang X (2022) A survey on epistemic (model) uncertainty in supervised learning: recent advances and applications. Neurocomputing 489:449–465