Probabilistic Bayesian Neural Networks for Efficient Inference

Md Ishak
Department of Electrical and Computer Engineering
Wayne State University
Detroit, MI, USA
md.ishak@wayne.edu

Mohammed Alawad
Department of Electrical and Computer Engineering
Wayne State University
Detroit, MI, USA
alawad@wayne.edu

ABSTRACT

Bayesian Neural Networks (BNNs) offer a principled framework for modeling uncertainty in deep learning tasks. However, conventional BNNs often suffer from high computational complexity and parameter overhead. In this paper, we propose a novel approach, termed Probabilistic BNN (ProbBNN), which leverages probabilistic computing principles to streamline the inference process. Unlike traditional deterministic approaches, ProbBNN represents inputs and parameters as random variables governed by probability distributions, allowing for the propagation of uncertainty throughout the network. We employ Gaussian Mixture Models (GMMs) to represent the parameters of each neuron or convolutional kernel, enabling efficient encoding and processing of uncertainty. Our approach simplifies the inference process by replacing complex deterministic computations with lightweight probabilistic operations, resulting in reduced computational complexity and improved scalability. Experimental results demonstrate the effectiveness of ProbBNN in achieving competitive accuracy to traditional BNNs while significantly reducing the number of parameters. The transition to ProbBNN yields a reduction of two orders of magnitude in the number of parameters compared to baseline approaches, making our approach promising for deployment in resource-constrained applications such as edge computing and IoT devices.

CCS CONCEPTS

• **Computing methodologies → Artificial intelligence**; • **Theory of computation → Probabilistic computation**.

KEYWORDS

Bayesian neural network, probabilistic computing, Gaussian Mixture Model

ACM Reference Format:
Md Ishak and Mohammed Alawad. 2024. Probabilistic Bayesian Neural Networks for Efficient Inference. In *Great Lakes Symposium on VLSI 2024 (GLSVLSI '24), June 12–14, 2024, Clearwater, FL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649476.3658740

1 INTRODUCTION

Bayesian Neural Networks (BNNs) [9] have garnered considerable attention in recent years for their ability to provide uncertainty quantification and robust decision-making, particularly in applications where understanding the reliability of predictions is crucial. Unlike traditional deterministic neural networks, which yield point estimates of parameters, BNNs treat model parameters as random variables governed by probability distributions. This probabilistic framework enables BNNs to capture uncertainty inherent in the data and model, offering insights into the reliability of predictions and enhancing model interpretability [14, 17] [11].

While BNNs hold great promise, their adoption is often hindered by computational complexity, especially in resource-constrained environments such as edge devices and IoT devices [10]. The inference process in BNNs typically involves computationally intensive operations, such as sampling from posterior distributions and feed-forward operations. During inference, BNNs often sample multiple neural network instances from the posterior distribution, denoted as T, to capture input data uncertainty. T can exceed 10 and, in certain scenarios, even surpass 100 [12]. Furthermore, the large number of parameters in traditional BNNs exacerbates these computational challenges, often leading to increased memory and processing requirements. The increase in parameter counts stems from the incorporation of mean and variance parameters for each weight in BNNs, a characteristic unique to probabilistic models, effectively doubling the total number of parameters compared to deterministic neural networks. This necessitates the development of more efficient and scalable approaches to Bayesian inference in neural networks, which can accommodate the growing demands of modern machine learning tasks.

Several approaches have been proposed to enhance the efficiency of BNN inference, addressing the computational complexity inherent in these models. Some of these approaches focus on optimizing the sampling process, aiming to improve the efficiency of posterior sampling techniques [4, 5]. Others target the dataflow process, optimizing the flow of data through the network to reduce computational overhead [13, 20]. Additionally, various architectural strategies have been explored, ranging from traditional FPGA implementations [7, 12] to emerging technologies [6, 10]. Despite these advancements, challenges persist in achieving optimal efficiency. Scalability of inference processes remains a concern, particularly when dealing with large-scale models and datasets. Furthermore, managing computational resources efficiently poses a significant challenge, especially in resource-constrained environments such as edge devices and IoT devices. Stochastic Computing-Based BNN (StocBNN) stands out as a promising approach to streamline the inference phase [12]. It leverages the stochastic computing methodology [1] to streamline the inference phase. In StocBNN, inputs and weights are represented in bitstream format, simplifying multiplication operations and reducing computational overhead. While offering advantages in terms of inference efficiency, StocBNN does not directly address the challenge of managing the large number of parameters inherent in probabilistic models. This characteristic can hinder its scalability and practical applicability, especially when dealing with complex models or resource-constrained environments.

In this paper, we propose Probabilistic Bayesian Neural Networks (ProbBNN), a novel approach that leverages probabilistic computing principles [2, 3] to streamline inference in BNNs. Traditional deterministic approaches often struggle with the computational complexity and memory demands associated with inference in BNNs. In ProbBNN, we represent both inputs and parameters as random variables governed by probability distributions, enabling the propagation of uncertainty throughout the network. By adopting this probabilistic representation, ProbBNN offers a more efficient and scalable alternative to conventional deterministic methods.

ProbBNN introduces several key innovations to enhance the efficiency and effectiveness of Bayesian inference in neural networks. Firstly, we employ Gaussian Mixture Models (GMMs) [18] to represent parameters in both fully connected and convolutional layers. This representation allows ProbBNN to capture the underlying distributions of weights more effectively with a smaller set of mixture components, thereby mitigating the challenges associated with the large number of parameters inherent in BNNs without sacrificing accuracy. Secondly, ProbBNN simplifies the inference process by replacing complex deterministic computations with lightweight probabilistic operations. This paradigm shift reduces computational complexity and resource requirements, making ProbBNN well-suited for deployment in resource-constrained environments.

The remainder of this paper is organized as follows: Section 2 provides an overview of BNNs, stochastic BNN, and probabilistic computing. In Section 3, we introduce the application of probabilistic computing in BNNs. We evaluate the performance of our approach on benchmark datasets and discuss experimental results in Section 4. Finally, we conclude the paper in Section 5 with a summary of our contributions and directions for future research.

2 BACKGROUND

2.1 Bayesian Neural Networks (BNNs)

BNNs present a departure from traditional deterministic neural networks by adopting a probabilistic approach to modeling. In deterministic neural networks, parameters such as weights (W) are treated as fixed values, learned through optimization techniques like gradient descent. However, in BNNs, these parameters are regarded as random variables governed by probability distributions, allowing for the representation of uncertainty in model predictions.

Mathematically, while a deterministic neural network computes the output y given input x as $y = f(x; W, b)$, where f represents the network's architecture and W and b are fixed parameters, a BNN computes the posterior distribution $p(W, b|X, y)$ over parameters given observed data X and labels y.

The process of training a BNN involves Bayesian inference to estimate the posterior distribution $p(W, b|X, y)$. This is typically done using techniques such as Markov Chain Monte Carlo (MCMC) or Variational Inference (VI). In contrast, training a deterministic neural network involves optimizing a loss function to find the optimal values of parameters W and b that minimize the discrepancy between predicted and actual outputs.

One of the main advantages of BNNs over deterministic neural networks is their ability to quantify uncertainty in predictions. By representing parameters as distributions, BNNs can capture uncertainty inherent in the data or model. For instance, instead of providing a single point estimate of the output, a BNN produces a distribution over possible outputs, indicating the model's uncertainty about the prediction. This uncertainty quantification is particularly valuable in safety-critical applications such as medical diagnosis or autonomous driving. Additionally, BNNs offer built-in regularization through the use of prior distributions over parameters. By incorporating prior knowledge about the parameters' distribution, BNNs can prevent overfitting and improve generalization performance, especially in scenarios with limited data [8]. In contrast, deterministic neural networks rely on explicit regularization techniques such as dropout or weight decay to prevent overfitting.

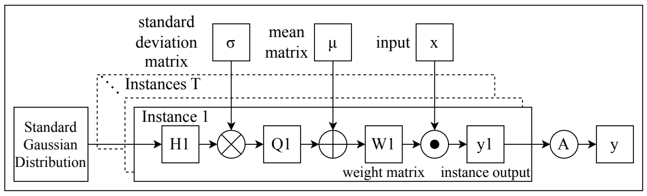

Figure 1: Single-layer digital BNN inference dataflow.

Digital BNNs (DigtBNNs), shown in Figure 1, encounter challenges primarily related to computation complexity and the sheer number of parameters. The computation complexity arises from the need to perform high-dimensional matrix multiplications, which can be resource-intensive, particularly for deep architectures. Running multiple instances of DigtBNNs is often necessary to achieve robustness and account for inherent variability in the training process, further amplifying computational demands and resource requirements. Moreover, the significant number of parameters, each requiring precision representation, contributes to memory and computational overheads and necessitates large storage capacities and computational resources. Balancing model complexity with computational efficiency poses a key challenge in digital BNN design. It requires innovative solutions to mitigate these complexities while preserving model accuracy and performance.

2.2 Stochastic Bayesian Neural Networks

Stochastic Computing (SC)-Based Bayesian Neural Networks, as introduced in [12] and shown in Figure 2, leverage the SC methodology for the inference phase of BNNs. In this paradigm, inputs and weights are encoded in bitstream format, enabling their direct involvement in the inference process. Notably, multiplication operations are simplified to AND operations, thus streamlining the feed-forward propagation compared to traditional approaches. Gaussian random numbers, pivotal in BNNs, are represented using a 0-1 bitstream format, and a simplified Gaussian Random Number

Generator (GRNG) is introduced to facilitate this representation. This not only demonstrates the feasibility of the representation but also elucidates the dataflow during feed-forward propagation in StocBNN.

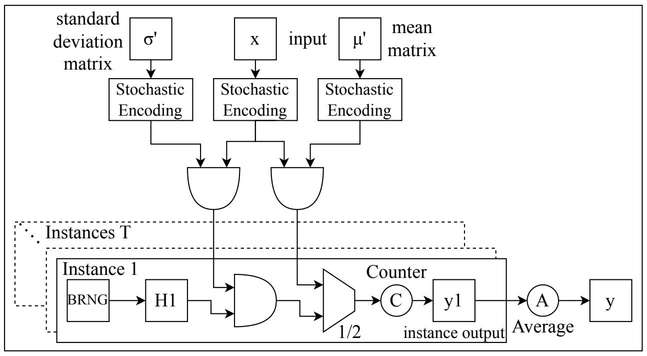

Figure 2: Single-layer stochastic BNN inference dataflow.

StocBNN addresses both weight sampling and feedforward propagation challenges in BNNs. For weight sampling, the CLT (Central Limit Theorem)-based GRNG employs a sequence of binary random number generators (BRNGs) to produce binary outputs. These outputs undergo CLT transformation, resulting in Gaussian random numbers sampled from a standard Gaussian distribution. The process involves utilizing a linear feedback shift register (LFSR) for implementing BRNGs, followed by a parallel counter to convert the output into digital Gaussian random numbers based on CLT. To overcome these challenges, the concept of bitstream format Gaussian random numbers is introduced. These bitstreams enable the efficient flow of Gaussian random numbers through the BNN's feed-forward propagation, thus bypassing the need for complex transformations and significantly reducing hardware resource consumption.

In the feed-forward propagation phase, Gaussian random numbers in bitstream format simplify the realization of the GRNG. Each element of the output, computed through matrix-vector multiplication, is determined by a series of binary random numbers. These binary random numbers, representing Gaussian random numbers in the bitstream domain, seamlessly participate in the computation. Through minor transformations, such as division by two and the use of multiplexers (MUXs), both multiplication and addition operations in BNNs are realized using the SC method. Consequently, StocBNN achieves efficient inference while maintaining the integrity of the computation, demonstrating its potential for practical implementation in neural network applications.

Despite its advantages in simplifying computation and reducing hardware resource consumption, StocBNN faces challenges related to achieving lower accuracy compared to conventional BNN approaches. Additionally, while it streamlines feed-forward propagation and simplifies multiplication operations, it does not inherently reduce the number of parameters in BNNs. Therefore, balancing computational efficiency with maintaining high accuracy remains a key challenge for StocBNN adoption.

2.3 Probabilistic Computing in Deep Learning

Probabilistic computing, as introduced in [2], is rooted in the theory of probabilistic convolution, which asserts that the addition of two random variables equates to the convolution of their respective probability density functions (PDFs). This computing paradigm fundamentally transforms how algorithms process inputs, initially encoding them probabilistically to generate a diverse ensemble of random samples. These samples then undergo lightweight operations, such as simple additions, to generate new random samples, which are subsequently decoded probabilistically to derive final results. In [3], this methodology was extended to encompass all layers—convolutional, ReLU, pooling, and fully connected—of deep learning models, treating both input values and kernel parameters as random variables governed by probability distributions, see Figure 3. This unified approach enables all computations to be conducted within the probabilistic domain, replacing complex deterministic computations with lightweight probabilistic operations, thereby enhancing computational efficiency in deep learning tasks.

Unlike traditional deterministic computing, probabilistic computing, as detailed in [2, 3], operates on the premise that inputs and operations are governed by probability distributions rather than fixed precision. This paradigm shift not only facilitates the representation of uncertainty in data and model parameters but also enhances robustness against noise and variability in real-world data. By eliminating the need for discretization and precision scaling, common challenges in deterministic computations, probabilistic computing streamlines the computational process and reduces resource requirements. Moreover, lightweight operations, such as simple additions, can be performed directly on random samples, further reducing computational complexity. In contrast to stochastic computing, which represents inputs and weights as bitstreams, probabilistic computing offers a more flexible and expressive framework for modeling uncertainty and performing probabilistic inference. While stochastic computing simplifies arithmetic operations to bitwise operations, probabilistic computing allows for a nuanced treatment of uncertainty, enabling richer representations of data distributions and more accurate inference. Overall, probabilistic computing presents a promising avenue for enhancing the efficiency and robustness of deep learning models, offering distinct advantages over traditional deterministic computing and stochastic computing approaches.

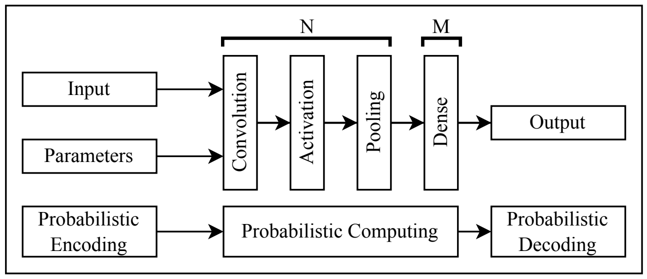

Figure 3: Probabilistic computing diagram, where N is the number of convolution layers and M is the number of fully connected layers.

3 PROBABILISTIC BAYESIAN NEURAL NETWORKS

BNNs offer a probabilistic framework for modeling uncertainty in deep learning tasks, enabling robust inference and decision-making. However, traditional BNN inference methods often encounter challenges related to data sampling and computational complexity. In this section, we explore how probabilistic computing, specifically applied in ProbBNN, addresses these issues by leveraging the principles of probabilistic convolution and lightweight probabilistic operations. By treating inputs and parameters as random variables governed by probability distributions, ProbBNN enhances the efficiency and accuracy of BNN inference, offering a promising solution to the limitations of deterministic and stochastic computing approaches.

3.1 Gaussian Mixture Models for Parameter Representation

In our proposed approach, we introduce Gaussian Mixture Models (GMMs) as a novel method for representing the parameters of each neuron in fully connected layers and each kernel in convolutional layers within BNNs. Unlike traditional BNNs, where each weight is represented by a mean and standard deviation, our method utilizes GMMs to capture the underlying distribution of parameters more effectively.

In a fully connected layer, let W represent the weights of a neuron, which typically consist of N individual weight parameters. Rather than representing each weight independently, we model the distribution of weights using a GMM with C components:

$$W \sim \sum_{i=1}^{C} \pi_i \mathcal{N}(\mu_i, \sigma_i) \tag{1}$$

where π_i denotes the mixing coefficient of the i-th component, μ_i represents the mean vector, and σ_i denotes the covariance matrix of the i-th Gaussian component. Similarly, in convolutional layers, each kernel is represented by a GMM with C components:

$$K \sim \sum_{i=1}^{C} \pi_i \mathcal{N}(\mu_i, \sigma_i) \tag{2}$$

where K represents the kernel, and π_i, μ_i, and σ_i have the same interpretations as in the fully connected layer case. The number of components C in each GMM is determined through an optimization process, where we aim to find the optimal trade-off between model complexity and representational power. By carefully selecting C, we can effectively capture the underlying distribution of parameters while minimizing redundancy and overfitting.

In this paper, we employ a grid search approach for optimizing the number of components in the GMMs used for parameter representation. While techniques such as model selection and component pruning can also be utilized for this purpose, they are not specifically employed in our study. The decision to use grid search is based on its simplicity and effectiveness in exploring a range of hyperparameters to find the optimal configuration for our model. However, future research could explore the integration of more advanced optimization techniques to further enhance the efficiency and performance of our proposed approach.

3.2 Probabilistic Bayesian Neural Network Inference

For our proposed approach, we depart from the deterministic computing paradigm commonly employed in conventional BNNs. Instead, we leverage probabilistic theory, particularly the principles of probabilistic convolution, to perform inference computations. Unlike traditional BNNs, which rely on deterministic operations such as multiply-accumulate (MAC) for inference, our approach utilizes addition-based operations.

Probabilistic convolution theory provides a framework for performing convolution operations in a probabilistic domain, where input and kernel values are treated as random variables governed by probability distributions. In our context, we extend this theory to the inference computations within the BNN architecture. Rather than computing point estimates of outputs using deterministic operations, we compute distributions over outputs by adding random samples generated following the GMM distribution for inputs and weights. This process continues through each layer of the network until the final layer, where a decoder is used to convert the resultant random samples into real values, representing the output of the BNN, denoted as "y" , as sown in Figure 4.

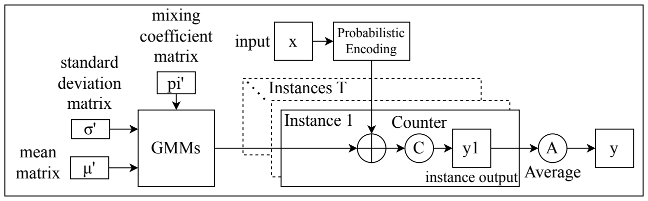

Figure 4: Single-layer probabilistic BNN inference dataflow.

By using probabilistic convolution, we maintain the probabilistic nature of BNNs throughout the inference process, allowing us to capture uncertainty in model predictions and propagate uncertainty through the network. Additionally, by employing addition-based operations instead of MAC operations, we simplify the computational complexity of the inference process, potentially leading to improved efficiency and scalability.

Furthermore, the principles of probabilistic convolution theory can also be extended to fully connected operations in the BNN. Instead of performing traditional matrix-vector multiplications, random samples generated from GMM distributions can be added together to compute the outputs of fully connected layers. This approach ensures that uncertainty is properly accounted for throughout all layers of the BNN architecture, contributing to more robust and accurate inference results.

4 EXPERIMENTAL SETUP AND RESULTS

Our experimentation is conducted on an Intel Xeon W-2295 machine operating at 3.00 GHz, running Ubuntu 22.04, and equipped with an Nvidia RTX A4000 GPU boasting 16 GB GPU memory.

In this study, our objective is to enhance the inference performance of BNNs using the proposed probabilistic computing method.

Table 1: Accuracy comparison between digital domain BNN (DigtBNN), SC domain BNN (StocBNN), and Probabilistic domain (ProbBNN)

Model	Dataset	DigtBNN	StocBNN			ProbBNN		
			32	64	128	32	64	128
4-FC	MNIST	98.29 %	97.18 %	97.20 %	97.51 %	97.42 %	97.88 %	98.21 %
	Fashion-MNIST	90.02 %	87.64 %	87.84 %	88.13 %	87.86 %	88.21 %	89.16 %
LeNet-5	MNIST	99.25 %	98.39 %	99.00 %	99.15 %	98.97 %	99.16 %	99.23 %
	Fashion-MNIST	99.36 %	99.14 %	99.28 %	99.34 %	99.21 %	99.34 %	99.36 %

Our experimental setup revolves around three main aspects: leveraging GMMs for model parameter representation, evaluating learning accuracy through software simulations of the proposed approach (ProbBNN), and comparing it with conventional digital BNNs (DigtBNN) and Stochastic BNNs (StocBNN) employing bit streams. Additionally, we explore the efficiency of the proposed approach concerning the number of parameters of BNN models. Experiments are conducted on well-known datasets, including MNIST [16] and Fashion-MNIST [21], to assess learning accuracy in classification tasks. Using Python, we maintain consistency by setting the neural network count T to 100. DigtBNN, implemented with 32-bit floating-point number format. The models are evaluated with a 784-200-200-10 configuration structure (4-FC), where all layers are fully connected. The BNN is pretrained using Edward [19], with ReLU activation functions. Experimental results on the 4-FC structure, employing a total of 10,000 test data, demonstrate a significant increase in testing accuracy with longer random sample stream lengths in our proposed ProbBNN and bitstream lengths in StochBNN. With a stream length of 128, ProbBNN outperforms StochBNN, with only a slight decrease in accuracy compared to DigtBNN, approximately 0.08% in MNIST and 0.96% in Fashion-MNIST. Moreover, we evaluate the efficiency of ProbBNN in CNN settings using the LeNet-5 [15] architecture. Our results show that ProbBNN achieves nearly identical accuracies compared to DigtBNN. Table 1 presents the accuracy comparisons among DigtBNN, StocBNN with bitstream lengths of 32, 64, and 128, and ProbBNN with a random sample stream lengths of 32, 64, and 128.

In our study, we rigorously assess the effectiveness of GMMs in approximating the PDF of sets of parameters, particularly those associated with neurons in feedforward NNs or kernels in CNNs. Through meticulous experimentation, we systematically vary the number of components in the GMMs to ascertain their ability to accurately represent these weight distributions. Figure 5 illustrates the approximation of neuron parameters in the 4FC layer, focusing on a neuron in the first hidden layer with a higher number of weights compared to other layers. GMMs with varying numbers of components are utilized to approximate these parameters, showcasing the efficiency of different component counts in the approximation process. Moreover, Figure 6 demonstrates how GMMs with just two components can effectively approximate a 5x5 kernel within the CNN model, highlighting the efficient representation of kernel parameters using a simplified GMM approach. Surprisingly, our analyses unveil that even with a conservative number of components GMMs exhibit remarkable proficiency in capturing the intricate structure of the parameters. This observed efficacy can be attributed, in part, to the nuanced interplay between model complexity and the nature of the tasks and BNN architectures examined in our investigation. Despite the relatively straightforward nature of the tasks and architectures, the robust performance of GMMs underscores their versatility and computational efficiency as indispensable tools for probabilistic modeling in neural network applications. These findings underscore the importance of deploying probabilistic techniques like GMMs to streamline model representation and inference, particularly in scenarios where resource constraints or scalability concerns are prevalent.

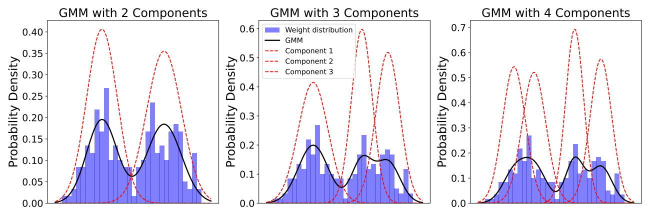

Figure 5: GMMs with a varying number of components for a neuron in FC model with 784 parameters.

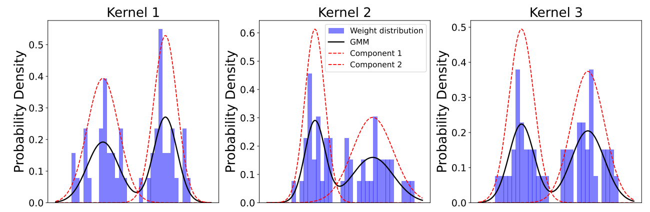

Figure 6: 2-component GMMs for three different (5x5) kernels in LeNet-5.

In our comparison between the proposed ProbBNN with DigtBNN and StocBNN, it's crucial to examine how the number of parameters is determined for each approach and the specific model considered. DigtBNN typically employs fixed-precision representations of weights, resulting in a straightforward parameter calculation based on the network architecture's size. For instance, in a 784-200-200-10 feedforward network, this results in 397,600 parameters, with each weight represented by a Gaussian distribution having two parameters for mean (μ) and variance (σ). Similarly, StocBNN utilizes stochastic computing with bit streams, leading to the same number of parameters as DigtBNN due to similar weight representations, where each parameter is represented by a Gaussian distribution with two parameters. In contrast, ProbBNN introduces a novel approach where each neuron's weights are represented by a GMM, significantly reducing the parameter count. Determining the number of parameters in ProbBNN involves considering the number of GMMS required to represent the weights of each neuron in the network. For example, in a 784-200-200-10 configuration, the total number of GMMs needed is determined by the number of neurons in each layer and the chosen representation scheme. In our experiments, we use grid search hyperparameter optimization to determine the number of components for GMMs and found that a modest number of GMMs are sufficient to accurately capture the underlying weight distributions. Specifically, we used four components per mixture for neurons with 784 weights in the first hidden layer and two components per mixture for neurons with 200 weights in the second hidden layer and output layer.

Additionally, each GMM component requires a weight or mixing coefficient to represent the relative contribution of each component to the overall mixture. Therefore, for each GMM component, we have three parameters: mean, variance, and mixing coefficient. This leads to a substantial reduction in the total number of parameters, with ProbBNN requiring only 3,660 parameters compared to the much larger parameter count in DigtBNN and StocBNN. Following the same approach, we calculate the number of parameters for the LeNet-5 model. For DigtBNN and StocBNN, the total number of weights is 118,940. For the proposed ProbBNN, we utilize two-component GMMs for each filter in the convolutional layers. Additionally, we employ three-component GMMs for each neuron in the first fully connected layer and two-component GMMs for each neuron in the second and output fully connected layers. The total number of parameters in ProbBNN is 1,776. This reduction underscores the efficiency and effectiveness of the proposed probabilistic approach in streamlining BNN architectures while maintaining modeling accuracy.

5 CONCLUSIONS

In conclusion, this research introduces a novel approach to realize BNNs by leveraging probabilistic convolution theory. We propose Probabilistic BNNs, where both inputs and parameters are treated as random variables governed by probability distributions. By representing weights using GMMs and performing inference computations in the probabilistic domain, we maintain the probabilistic nature of the model while simplifying the computational complexity of the inference process. Our experimental results demonstrate the effectiveness of ProbBNNs in reducing the number of parameters while maintaining modeling accuracy, offering a promising avenue for enhancing the efficiency and robustness of BNNs in various applications. Additionally, we highlight the challenges and opportunities for future research, including further optimization of hyperparameters and exploration of probabilistic computing techniques for other neural network architectures.

REFERENCES

[1] Armin Alaghi and John P. Hayes. 2013. Survey of Stochastic Computing. *ACM Trans. Embed. Comput. Syst.* 12, 2s, Article 92 (may 2013), 19 pages. https://doi.org/10.1145/2465787.2465794

[2] Mohammed Alawad, Yu Bai, Ronald F. DeMara, and Mingjie Lin. 2019. Robust and Large-Scale Convolution through Stochastic-Based Processing without Multipliers. *IEEE Transactions on Emerging Topics in Computing* 7, 1 (2019), 80–97. https://doi.org/10.1109/TETC.2016.2601220

[3] Mohammed Alawad and mingjie lin. 2018. Scalable FPGA Accelerator for Deep Convolutional Neural Networks with Stochastic Streaming. *IEEE Transactions on Multi-Scale Computing Systems* 4, 4 (2018), 888–899. https://doi.org/10.1109/TMSCS.2018.2886266

[4] Hiromitsu Awano and Masanori Hashimoto. 2020. BYNQNet: Bayesian Neural Network with Quadratic Activations for Sampling-Free Uncertainty Estimation on FPGA. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1402–1407. https://doi.org/10.23919/DATE48585.2020.9116302

[5] Hiromitsu Awano and Masanori Hashimoto. 2023. B2N2: Resource efficient Bayesian neural network accelerator using Bernoulli sampler on FPGA. *Integration* 89 (2023), 1–8. https://doi.org/10.1016/j.vlsi.2022.11.005

[6] Richard Dorrance, Deepak Dasalukunte, Hechen Wang, Renzhi Liu, and Brent Carlton. 2022. Energy Efficient BNN Accelerator using CiM and a Time-Interleaved Hadamard Digital GRNG in 22nm CMOS. In *2022 IEEE Asian Solid-State Circuits Conference (A-SSCC)*. 2–4. https://doi.org/10.1109/A-SSCC56115.2022.9980539

[7] Hongxiang Fan, Martin Ferianc, Zhiqiang Que, Shuanglong Liu, Xinyu Niu, Miguel R. D. Rodrigues, and Wayne Luk. 2022. FPGA-Based Acceleration for Bayesian Convolutional Neural Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 12 (2022), 5343–5356. https://doi.org/10.1109/TCAD.2022.3160948

[8] Yarin Gal and Zoubin Ghahramani. 2015. Bayesian convolutional neural networks with Bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158* (2015).

[9] Z Ghahramani. 2015. Probabilistic machine learning and artificial intelligence. *Nature* 521, 7553 (May 2015), 452–459. https://doi.org/10.1038/nature14541 On Probabilistic models.

[10] Huiyi Gu, Xiaotao Jia, Yuhao Liu, Jianlei Yang, Xueyan Wang, Youguang Zhang, Sorin Dan Cotofana, and Weisheng Zhao. 2023. CiM-BNN:Computing-in-MRAM Architecture for Stochastic Computing Based Bayesian Neural Network. *IEEE Transactions on Emerging Topics in Computing* (2023), 1–11. https://doi.org/10.1109/TETC.2023.3317136

[11] Ali Haghani Hyoshin Park and Xin Zhang. 2016. Interpretation of Bayesian neural networks for predicting the duration of detected incidents. *Journal of Intelligent Transportation Systems* 20, 4 (2016), 385–400. https://doi.org/10.1080/15472450.2015.1082428 arXiv:https://doi.org/10.1080/15472450.2015.1082428

[12] Xiaotao Jia, Huiyi Gu, Yuhao Liu, Jianlei Yang, Xueyan Wang, Weitao Pan, Youguang Zhang, Sorin Cotofana, and Weisheng Zhao. 2023. An Energy-Efficient Bayesian Neural Network Implementation Using Stochastic Computing Method. *IEEE Transactions on Neural Networks and Learning Systems* (2023), 1–11. https://doi.org/10.1109/TNNLS.2023.3265533

[13] Xiaotao Jia, Jianlei Yang, Runze Liu, Xueyan Wang, Sorin Dan Cotofana, and Weisheng Zhao. 2021. Efficient Computation Reduction in Bayesian Neural Networks Through Feature Decomposition and Memorization. *IEEE Transactions on Neural Networks and Learning Systems* 32, 4 (2021), 1703–1712. https://doi.org/10.1109/TNNLS.2020.2987760

[14] Alex Kendall and Yarin Gal. 2017. What uncertainties do we need in Bayesian deep learning for computer vision?. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 5580–5590.

[15] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[16] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010). http://yann.lecun.com/exdb/mnist/

[17] Cedrique Rovile Njieutcheu Tassi. 2020. Bayesian Convolutional Neural Network: Robustly Quantify Uncertainty for Misclassifications Detection. In *Pattern Recognition and Artificial Intelligence*, Chawki Djeddi, Akhtar Jamil, and Imran Siddiqi (Eds.). Springer International Publishing, Cham, 118–132.

[18] Douglas A Reynolds et al. 2009. Gaussian mixture models. *Encyclopedia of biometrics* 741, 659-663 (2009).

[19] Dustin Tran, Matthew D Hoffman, Rif A Saurous, Eugene Brevdo, Kevin Murphy, and David M Blei. 2017. Deep probabilistic programming. *arXiv preprint arXiv:1701.03757* (2017).

[20] Qiyu Wan and Xin Fu. 2020. Fast-BCNN: Massive Neuron Skipping in Bayesian Convolutional Neural Networks. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 229–240. https://doi.org/10.1109/MICRO50266.2020.00030

[21] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).