When Monte-Carlo Dropout Meets Multi-Exit: Optimizing Bayesian Neural Networks on FPGA

Hongxiang Fan†
Samsung AI Center & University of Cambridge
Cambridge, UK
hongxiangfan@ieee.org

Mark Chen
Department of Computing Imperial College London
London, UK
hao.chen20@imperial.ac.uk

Liam Castelli
Department of Computing Imperial College London
London, UK
castelliliam@gmail.com

Zhiqiang Que
Department of Computing Imperial College London
London, UK
z.que@imperial.ac.uk

He Li
Department of Electronics and Engineering Southeast University
Nanjing, China
helix@seu.edu.cn

Kenneth Long
Department of Physics Imperial College London
London, UK
k.long@imperial.ac.uk

Wayne Luk
Department of Computing Imperial College London
London, UK
w.luk@imperial.ac.uk

***Abstract*—Bayesian Neural Networks (BayesNNs) have demonstrated their capability of providing calibrated prediction for safety-critical applications such as medical imaging and autonomous driving. However, the high algorithmic complexity and the poor hardware performance of BayesNNs hinder their deployment in real-life applications. To bridge this gap, this paper proposes a novel multi-exit Monte-Carlo Dropout (MCD)-based BayesNN that achieves well-calibrated predictions with low algorithmic complexity. To further reduce the barrier to adopting BayesNNs, we propose a transformation framework that can generate FPGA-based accelerators for multi-exit MCD-based BayesNNs. Several novel optimization techniques are introduced to improve hardware performance. Our experiments demonstrate that our auto-generated accelerator achieves higher energy efficiency than CPU, GPU, and other state-of-the-art hardware implementations. Our code is publicly available at: https://github.com/os-hxfan/MCME_FPGA_Acc.git**

***Index Terms*—Bayesian Neural Networks, Multi-Exit Optimization, Field Programmable Gate Array (FPGA)**

I. Introduction

Deep neural networks (DNNs) have become a frontier of artificial intelligence, with a variety of applications in many domains including computer vision, natural language understanding, medical image processing and scientific data analysis [1]. However, conventional deep neural networks have an important drawback: they behave like black boxes so can neither explain how the answers are obtained, nor provide an estimate of their confidence in their correctness. Bayesian Neural Networks (BayesNNs) [2] have been introduced to address the lack of confidence estimation of conventional deep neural networks, or non-Bayesian NNs (non-BayesNNs). The uncertainty-aware feature of BayesNNs makes them more resilient to risks caused by over-confident prediction of non-BayesNNs.

Nevertheless, there are two major challenges in deploying BayesNNs in real-life applications. First, the high dimensionality of modern BayesNNs significantly increases their algorithmic complexity, making exact Bayesian inference intractable. Although various approximation approaches, such as variational inference [3] and Monte-Carlo Dropout (MCD) [4], have been introduced to reduce computational overhead, these approaches perform worse in terms of uncertainty quality and calibration ability [5] than traditional deep ensembles that consist of multiple DNNs. Second, even with approximations, the computational and memory demands of BayesNNs are still much higher than those of non-BayesNNs due to Monte-Carlo (MC) sampling, hindering their deployment in demanding applications, especially those with real-time requirements. While there is extensive research on hardware acceleration for deep learning algorithms, most existing efforts focus on domain-specific hardware [6]–[8] or design automation tools [9], [10] for non-BayesNNs such as convolutional NNs (CNNs) [11], [12] and long short-term memory (LSTM) [13]. Hence there is an urgent need for publicly accessible hardware acceleration for BayesNNs.

To address the first challenge, this paper proposes a novel multi-exit MCD-based BayesNN. Compared with traditional MCD-based BayesNNs, our method is able to provide well-calibrated predictions. Also, the proposed multi-exit MCD-based BayesNNs are less computational and memory-intensive than conventional deep ensembles while possessing the flexibility to generate arbitrary numbers of MC samples. To overcome the second challenge, we propose to accelerate multi-exit MCD-based BayesNNs on FPGA. A transformation framework is introduced to generate high-performance FPGA-based accelerators for multi-exit MCD-based BayesNNs. With several novel optimizations such as spatial-temporal mapping and algorithm-hardware co-exploration, the generated accelerators achieve higher energy efficiency than previous hardware implementations. Moreover, this paper provides the first systematic study quantifying the benefits and the overheads of accelerated BayesNNs against their non-BayesNN counterparts. It would be of interest to DNN application builders to understand the trade-offs in deploying FPGA designs of BayesNNs to replace those implementing non-BayesNNs.

The contributions of this paper can be summarized as follows:

- A novel multi-exit MCD-based BayesNN with better calibration ability than conventional MCD-based BayesNN, and higher computational efficiency and flexibility over traditional deep ensembles.
- A design framework for transforming non-BayesNN models to multi-exit BayesNN hardware accelerators with high hardware performance and energy efficiency.
- Various optimization strategies including spatial-temporal mapping and algorithm-hardware co-exploration for performance improvement.

II. Background and Related Work

A. Bayesian Neural Networks

BayesNNs are able to achieve robustness against overfitting and to provide the estimation of their model uncertainty by means of Bayesian inference. Instead of capturing point-wise weight values like non-BayesNNs, BayesNNs are trained to learn the distribution of the weights. The Bayes rule is adopted in learning the distribution $p(w|D)$ for the weights w with respect to training data D. It is, however, computationally intractable to calculate the posterior

† Work was done while pursuing Ph.D. at Imperial College London.

distribution $p(w|D)$ analytically due to the high dimensionality of modern BayesNNs. To address this issue, various approximation methods have been introduced for BayesNNs [5]. Among these approaches, Monte Carlo Dropout (MCD) [4] is drawing attention as it provides an efficient way to estimate uncertainty, which is achieved by interpreting the dropout training of DNNs as approximate Bayesian inference for deep Gaussian processes.

MCD is implemented by applying a random filter-wise mask to the output feature maps of a layer i with F_i dimensional filters, which randomly drops out connections in a neural network. The values for the mask M_i adopt a Bernoulli distribution $p(M_i|p_i)$ with binary random variables (0 or 1) with probability p_i. The inference of MCD-based BayesNNs requires running multiple forward passes to generate different MC samples. In other words, the uncertainty estimation and calibration are achieved by feeding the same input through a BayesCNN S times, each time with a different set of sampled masks M. Dropouts have been employed in non-BayesNNs, typically during training. In contrast, for BasyesNNs, MCD-based dropout takes place during training as well as during inference.

B. Multi-Exit Network

Deep ensembles involve combining the predictions from multiple individual neural networks. They enable high prediction accuracy and high quality of calibration and uncertainty quantification [14]. However, training and using these networks can be prohibitively expensive. An alternative approach is to use a multi-exit architecture. By introducing intermediary classifiers before the final exit, multiple predictions can be obtained in a single pass. Using an equally weighted ensemble of the predictions from each of these exits can be shown to achieve accurate uncertainty estimation [15].

While some architectures like the Multi-Scale DenseNet [16] have been specifically designed for multi-exit, the most common approach to developing these architectures is to use a known powerful backbone architecture like ResNet, and then to attach intermediary classifiers at particular points [17], [18]. These exit points can be selected in a variety of ways by floating-point operation (FLOP) thresholds or semantic groupings of convolutional layers [17], [18].

C. Related Work

Much research has taken place on deep neural networks and their applications [1], and the use of FPGAs to accelerate deep neural networks. Representative work in this area includes energy-efficient CNN acceleration [7] and FPGA-based real-time AI cloud services [6]. There has also been significant research into design automation for deep neural networks. One example is the open source tool *hls4ml* supporting an automatic design flow involving high-level synthesis to promote low-power machine learning [10].

FPGA-based acceleration of BayesNNs has been reported recently [19]. One example of an early design is *Bynqnet*, an FPGA-based Bayesian neural network with quadratic activations for sampling-free uncertainty estimation [20]. Efficient FPGA implementations for 2D and 3D BayesNNs have been reported [21]. Another example is *VIBNN*, an FPGA-based accelerator that supports variational inference in BaynesNNs [22]. There is also research on algorithmic and hardware optimizations of BayesNNs, exploiting their structured sparsity and redundant computations [23]. In contrast to these approaches, this work proposes to accelerate multi-exit MCD-based BayesNNs on FPGA, achieving high energy efficiency and well-calibrated predictions.

III. Multi-Exit Meets MCD

As discussed in Section II, both multi-exit and MCD-based approaches are able to generate calibrated predictions, but they also have clear limitations. Although MCD-based methods provide an efficient approximation for BayesNNs, their predictive uncertainty and calibration ability have been demonstrated to be worse than deep ensembles [5]. The introduction of MCD layers after each convolution in vanilla MCD-based BayesNNs can hamper their predictive power, worsening both its accuracy and its uncertainty quantification [24].

In contrast, the multi-exit approach can be interpreted as deep ensembles with a shared backbone network [15], but it lacks flexibility when the calibration requires more predictive outputs than the number of exits. To alleviate the drawbacks of both of these individual approaches, we propose multi-exit MCD-based BayesNNs.

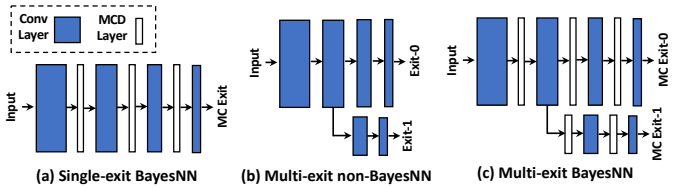

Fig. 1. Difference between a single-exit BayesNN, a multi-exit NN and a multi-ext BayesNN.

Figure 1 illustrates the difference between a vanilla MCD-based Bayesian NN, a multi-exit non-BayesNN, and a multi-exit MCD-based BayesNN. By introducing MCD layers in the multi-exit architecture, the proposed approach can achieve a similar level of calibration ability to deep ensembles. Moreover, multi-exit MCD-based BayesNNs can perform MC sampling by running the introduced MCD layers multiple times, which enables the ability to generate arbitrary numbers of MC samples. As discussed above, the usage of MCD layers after every convolution can introduce too much regularization into the network, leading to worse accuracy and worse uncertainty quantification [25]. Also, by placing MCD layers as close to each exit as possible, fewer computations are required since the non-Bayesian results can be cached and reused for different MC samples. Therefore, rather than adopting the fully MCD-based approach, we insert MCD layers starting from exits towards the input. The number of MCD layers is a hyper-parameter for optimization. We refer the layers without MCD applied as the non-Bayesian component.

To demonstrate that the multi-exit MCD-based BayesNN with both Bayesian and non-Bayesian components is still a mathematically valid approximation to BayesNNs, one can interpret the non-Bayesian component as a feature extractor. Therefore, given an M-exit architecture with inputs $\mathbf{X}$, our approach first maps the data from input space into feature space by using $f_i(X)$, where $f_i(.)$ denotes the feature extractor of each exit with $1 \leq i \leq M$. By replacing the $\mathbf{X}$ with $f_i(\mathbf{X})$ in the mathematical proof provided by [4], our multi-exit MCD-based BayesNN can be interpreted as the ensembles of approximated BayesNNs built upon the feature space.

Another benefit of multi-exit BayesNNs is the lower computational cost of generating MC samples than single-exit BayesNNs. Given that the floating-point operations (FLOPs) of the main body and all the exits are respectively $FLOP_{main}$ and $FLOP_{exit}$. As getting one MC sample needs to run the entire network in single-exit BayesNNs, the computational cost of running N_{sample} MC samples can be

formulated as:

$$N_{sample} \times (FLOP_{main} + FLOP_{exit}). \quad (1)$$

In contrast, the required FLOPs of an N_{exit} multi-exit BayesNN to get the same number of MC samples is:

$$FLOP_{main} + \frac{N_{sample}}{N_{exit}} \times FLOP_{exit}. \quad (2)$$

The reduction rate is given by dividing Equation 1 by Equation 2,

$$\frac{1+\alpha}{\frac{1}{N_{sample}} + \frac{\alpha}{N_{exit}}}, \quad (3)$$

where $\alpha = \frac{FLOP_{exit}}{FLOP_{main}}$. The reduction rate varies by different multi-exit architectures, depending on N_{sample}, N_{exit} and α.

Section II-B discusses the wide variety of possible methods in which multi-exit networks can be created and trained. In this work, the exit branches are placed according to the approach used in [17]. Semantic groupings are formed for each network, splitting the network architecture into "blocks" separated by pooling layers. An exit branch is then placed after each of these blocks. In order to allow for more direct validation of the work performed in this paper, the bidirectional distillation training method in [17] is used.

IV. Transformation Framework

A. Framework Overview

An overview of our transformation framework is presented in Figure 2. There are four phases in our proposed framework: *(1)* construction and optimization of multi-exit MCD-based BayesNNs, *(2)* spatial and temporal mapping optimization, *(3)* design space exploration for both algorithm and hardware design and *(4)* generation of BayesNN accelerators based on HLS (High-Level Synthesis).

Given the neural architecture of a non-BayesNN as the input, the first phase constructs a multi-exit MCD-based BayesNN by optimizing the multi-exit architecture, the number of MCD layers, and dropout rates. The second phase applies both temporal and spatial mappings to improve the hardware performance. The third phase involves algorithm and hardware co-exploration to optimize design parameters such as bitwidth and execution strategy. The last phase generates the corresponding HLS-based hardware accelerator. We adopt the design flow and HLS template of non-Bayesian layers from *HLS4ML*. To support the generation of BayesNN accelerators, we add the HLS-based implementation of MCD layers and *Keras*-to-HLS conversion into the design flow.

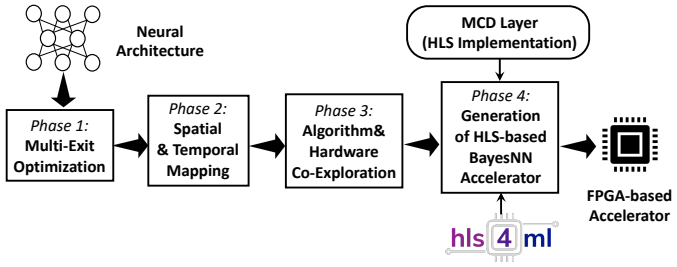

Fig. 2. Framework Overview.

B. Multi-exit Optimization: Phase 1

Since the number of exits is a design parameter in multi-exit BayesNNs, it presents a larger design space than the conventional BayesNNs. Given that a multi-exit BayesNN has N_{exit} exits, the number of forward passes N_{pass} required to produce the total number of MC samples N_{sample} is given by $N_{pass} = \frac{N_{sample}}{N_{exit}}$. The higher number of N_{exit} and N_{pass} may improve both the accuracy and calibration. However, it can degrade hardware performance due to the larger amount of computational and memory demands. As different applications and tasks may have different requirements for algorithm and hardware performance, we propose an optimization exploration flow as shown in Figure 3.

The optimization flow starts from the model construction of multi-exit BayesNNs given the input model architecture. By inserting N_{exit} exits and an MCD layer with dropout rate $P_{dropout}$, different multi-exit BayesNNs are constructed and trained on the target dataset. When the training finishes, we evaluate different metrics for multi-exit BayesNNs, including accuracy, calibration and the amount of floating-point operations (FLOPs).

Based on the evaluated performance, the design points that do not meet user constraints are filtered out. Then, according to the optimization priority, design space exploration is performed to find the optimal design configuration via grid search. The optimization priority can be based on accuracy, calibration and the amount of FLOPs. The final optimized design is fed into the next stage for hardware design generation.

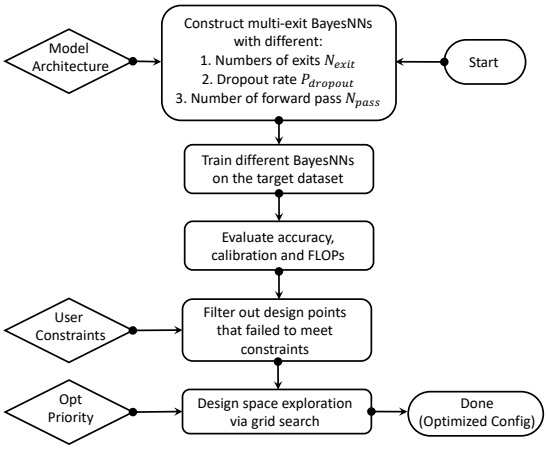

Fig. 3. Optimization flow.

C. Spatial and Temporal Mappings: Phase 2

The inference of Bayesian components requires multiple forward passes to obtain different MC samples. This Bayesian-related computation exhibits concurrency along the sampling dimension compared with conventional non-Bayesian NNs, enabling new parallelism strategies in hardware design. Therefore, we propose two mapping strategies, spatial and temporal mappings, to accelerate the Bayesian component of Multi-exit MCD-based BayesNNs, which are illustrated in Figure 4.

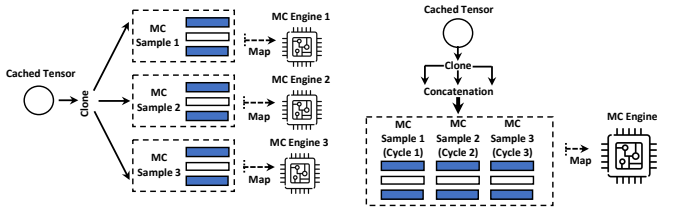

Fig. 4. Spatial and temporal mappings for Bayesian components.

In both mapping strategies, the tensor generated from the last non-Bayesian layer is cached and cloned into multiple copies. As shown in Figure 4(a), the spatial mapping deploys separate hardware MC Engines for different MC samples. Although spatial mapping effectively reduces latency by spatially parallelizing the sampling dimension, it also significantly increases resource use when the number of MC samples becomes large. To alleviate this issue, we propose temporal mapping that shares one MC Engine among multiple MC samples. As shown in Figure 4(b), the cloned copies are concatenated before feeding into the shared engine, which maps different MC samples one by one onto a single MC Engine. Our approach optimizes the mix of spatial and temporal mappings to meet different latency and resource constraints.

D. Algorithm and Hardware Co-Exploration: Phase 3

Our hardware accelerator contains different design parameters, such as the implementation strategy used in *HLS4ML*, the reuse factor specified for each layer, and the mapping strategy adopted for the Bayesian component. On the algorithmic side, given the input model architecture, there are several hyper-parameters that can be optimized, including the channel number and the bitwidth of activations and weights. We adopt grid search to optimize both algorithm and hardware design parameters with the requirement of not reducing the algorithmic performance compared to the default configurations. To reduce search costs, we experiment with heuristics such that the bitwidth is chosen from $\{4, 6, 8, 16\}$, and the channel number is selected from $\{C, \frac{C}{2}, \frac{C}{4}, \frac{C}{8}\}$ with C being the original number of channels. Users can also define other search space.

E. Generation of FPGA-based Accelerator: Phase 4

The generation of hardware accelerators is based on *HLS4ML* and our customized MCD design template. The generated HLS-based BayesNN accelerators can then be fed into Vivado-HLS for synthesis and implementation to get the final bitstream for onboard testing. The pseudocode of HLS-based implementation of MCD is presented in Algorithm 1. The HLS directive *HLS PIPELINE* is used to improve the overall performance. We cache the temporary result in the variable *temp*, before generating the final outputs. The hardware design receives the stream input data from the preceding layer, and produces stream outputs to the following layer. The dropout rate $P_{dropout}$ is a design parameter specified by the user at the beginning of running each model. A multiplexer is used to select either 0 or the result of the multiplication between inputs and dropout rate $P_{dropout}$. The control signal of the multiplexer is generated by comparing $P_{dropout}$ with *uniform_random*. To support the MCD layer with arbitrary $P_{dropout}$, a random number generator is used in our design to generate *uniform_random*.

Algorithm 1 Pseudocode of MCD layer

```
1: Input: input[dropout_size], keep_rate
2: Output: output[dropout_size]
3: for (i from 0 to dropout_size) do          ▷ #pragma PIPELINE
4:     temp = input[i]
5:     uniform_random = random_number_generator()
6:     if (uniform_random > keep_rate) then temp = 0
7:     output[i] = temp * keep_rate
```

V. Experiments and Evaluation

Our optimization framework is implemented in Python 3.8.12, PyTorch 1.11.0, and Keras 2.9.0. We use Vivado-HLS 2020.1 for hardware implementation. QKeras is used for quantization. The latency and resource consumption are obtained from C-synthesis reports provided by Vivado-HLS. Vivado 2020.1 is used to run place and route for the final designs. We set Xilinx Kintex XCKU115 as our target FPGA board. All the designs are optimized by our spatial-temporal mapping and algorithm-hardware co-exploration to ensure they can be fitted into the target platform.

A. Cost of Being Bayesian

The first experiment evaluates the hardware cost of supporting BayesNNs on FPGA. Compared with non-BayesNN designs, the hardware overhead of BayesNN accelerators comes from the use of MCD layers and the need to run multiple MC samples. To quantitatively investigate the cost, we evaluate the three BayesNNs on different datasets, i.e., *LetNet5* on MNIST, *ResNet-18* on CIFAR-10, and *VGG-11* on SVHN. As this experiment aims to evaluate the cost of being Bayesian, we use one exit on each model to eliminate the hardware overhead introduced by the multi-exit optimization. The custom configurations of these models, including quantization channel settings, are available in our open-sourced code. The results are presented in Figure 5.

To evaluate the resource overhead, we generate and synthesize the designs using temporal mapping (Section IV-C) with different numbers of MCD layers. The resource consumption of Block RAM (BRAM), DSP, Flip-Flop (FF) and LUT is shown on the left of Figure 5. The utilization of logic resources, including FF and LUT, shows an increasing trend when the number of MCD layers becomes larger. The increase of DSP is not significant, except for *Bayes-VGG11* which has an 8% increase with 7 MCD layers. As the design of the MCD layer does not require BRAM in the design, the BRAM consumption remains the same across different numbers of MCD layers on all three models. To measure the latency cost of MC sampling, we evaluate the designs with one MCD layer using different numbers of MC samples. To demonstrate the effect of spatial mapping optimization, we compare the latency of two implementations with and without spatial mapping. For the unoptimized version, we assume a single engine is used for multiple MC samples. The right of Figure 5 shows an increase in latency when the number of MC samples becomes large in the unoptimized implementations. In contrast, due to the parallelization of different MC samples, the latency cost of spatial mapping almost stays the same with the increasing number of MC samples, demonstrating the effectiveness of spatial mapping.

B. Effect of Multi-Exit BayesNNs

To demonstrate the advantage of multi-exit BayesNNs over the baseline approaches, we evaluate two commonly-used multi-exit models, *VGG19* and *ResNet18* for image classification. Cifar100 dataset, a curated subset of a larger dataset scraped from the web containing photo-realistic tiny 32×32 images with a single main object, is used in this experiment.

We compare four different implementations: *i)* Single-exit model with only one exit at the end of the network (SE). There is no MCD or Multi-Exit applied, which is the original implementation of both the *ResNet-18* and *VGG-19*. *ii)* MCD-based BayesNN without multi-exit (MCD). The MCD is only applied to the single exit of the network. *iii)* Multi-exit model without MCD (ME). We add one exit after each ResNet and VGG block to make multiple exits. *iv)* MCD-based BayesNN with multi-exit (MCD + ME). The MCD is applied to every exit of the network. Stochastic gradient descent (SGD) is used with a weight decay of 5×10^{-4}, an initial learning rate of 0.1 and a momentum of 0.9, along with a batch size of 64.

TABLE I
PERFORMANCE COMPARISON AMONG SE CNNS, MCD BAYESNNS, ME AND MCD-ME BAYESNNS WITH 32-BIT FLOATING POINT (FP32).

	ResNet18 (FP32)				VGG19 (FP32)			
	Acc-Opt		ECE-Opt		Acc-Opt		ECE-Opt	
	Accuracy	FLOPs	ECE	FLOPs	Accuracy	FLOPs	ECE	FLOPs
SE	0.752 ± 0.002	**1.00**	0.0840 ± 0.0008	1.00	0.693 ± 0.002	1.00	0.165 ± 0.006	1.00
MCD	0.758 ± 0.002	1.00	0.069 ± 0.001	1.00	0.696 ± 0.004	1.00	0.131 ± 0.006	1.00
ME	0.7719 ± 0.0006	1.026 ± 0.003	0.017 ± 0.002	1.026 ± 0.003	0.747 ± 0.002	**0.977**	0.025 ± 0.001	0.46 ± 0.05
MCD+ME (Ours)	$\mathbf{0.776 \pm 0.001}$	1.019 ± 0.004	$\mathbf{0.014 \pm 0.001}$	$\mathbf{0.672 \pm 0.003}$	$\mathbf{0.747 \pm 0.001}$	0.982	$\mathbf{0.017 \pm 0.001}$	$\mathbf{0.45 \pm 0.02}$

TABLE II
PERFORMANCE COMPARISON OF OUR FINAL FPGA DESIGNS WITH CPU, GPU, AND OTHER FPGA-BASED IMPLEMENTATIONS.

	CPU	**GPU**	**ASPLOS'18 [22]**	**DATE'20 [20]**	**DAC'21 [26]**	**TPDS'22 [23]**	**Our Work**
Platform	Intel Core i9-9900K	NVIDIA RTX 2080	Altera Cyclone V	Zynq XC7Z020	Arria 10 GX1150	Arria 10 GX1150	XCKU 115
Frequency (MHz)	3600	1545	213	200	225	220	181
Technology	14 nm	12 nm	28 nm	28 nm	20 nm	20 nm	20 nm
Power (W)	205	236	6.11	2.76	45.00	43.6	4.6
Latency (ms)	1.26	0.57	5.5	4.5	0.42	0.32	0.89
Energy Efficiency (J/Image)	0.258	0.134	0.033	0.012	0.019	0.014	0.004

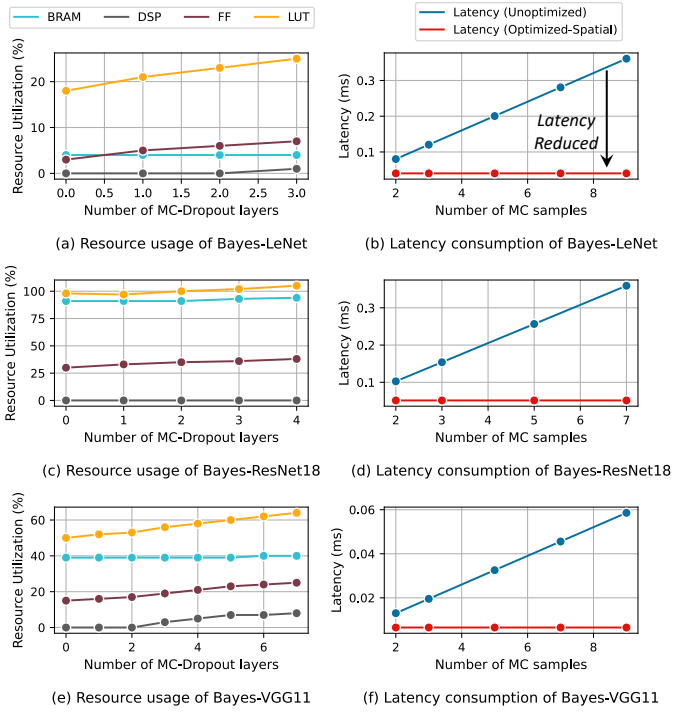

Fig. 5. Resource consumption and latency of Bayes-LeNet, Bayes-ResNet18 and Bayes-VGG11 with quantization and custom number of channels.

As discussed previously, the usage of too many dropout layers in a BayesNN can overregularize the network and adversely affect performance. However, there is no standard method to find the best balance between the level of dropout and calibration. Therefore, a small grid search is performed over the following dropout rates: 0.125, 0.25, 0.375, 0.5. Similarly, the confidence threshold which optimally balances the computational cost and the network performance is found through testing the same thresholds as in [18]: 0.1, 0.15, 0.25, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99, 0.999. It is noted that two sets of results from performing confidence-based exiting are calculated, using the predictions at each exit or the largest possible ensemble at each exit respectively. Each ensemble is an equally weighted average of the predictions from each exit, as in [15].

The grid search covers all combinations of the above two parameters, which is applied to the applicable networks. The predictions from each of the exits and the ensembles formed by averaging the results from each exit are calculated, alongside the predictions from confidence exiting. The best results are presented in Table I with the calibration captured by expected calibration error (ECE) [23]: a low value of ECE denotes a higher quality. As the dropout rate of MCD and the confidence threshold of multi-exit may affect both accuracy and calibration, two configurations for each implementation and model are reported: those that achieve the highest accuracy (*Acc-Opt*) and those with the lowest ECE (*Acc-ECE*). For each configuration, we also calculate the FLOPs as a fraction of the SE implementation.

On *ResNet18*, our approach, MCD + ME, improves the accuracy by 2.4% $\pm$0.002% with only 0.019 times more FLOPs compared with the SE implementation. Our method also shows higher accuracy than both MCD and ME implementations. In *Acc-ECE*, we achieve the lowest ECE and FLOPs among four implementations. A similar trend can also be observed in *VGG-19*. Moreover, our approach can match or outperform both of the methods individually, while costing a similar amount of FLOPs. The best model is able to massively reduce the ECE of the SE implementation by 0.148 ± 0.006, an improvement of almost 90%, while costing less than half the amount of FLOPs. These results show that multi-exit BayesNNs can lead to better calibrated and more powerful networks, while costing similar or fewer FLOPs.

C. Comparison with CPU, GPU, and FPGA implementations

To demonstrate the energy efficiency of our approach, we also compare it against CPU, GPU, and other FPGA-based implementations. The comparison uses MNIST dataset since it is the most common dataset across different work [20], [22], [23], [26]. As

TABLE III
POWER BREAKDOWN OF OUR FPGA-BASED ACCELERATOR.

	Dynamic (W)					**Static**	**Total**
	Clocking	Logic& Signal	BRAM	IO	DSP		
Used	0.374	1.359	0.422	0.998	0.191	1.299	4.6
Percentage	8%	30%	9%	21%	4%	28%	100%

both [22] and [20] do not support *Bayes-LeNet5*, we use their reported throughput (GOP/s) to estimate their performance on *Bayes-LeNet5*. The performance is obtained by using three MC samples. Both CPU and GPU performance are quoted from the vanilla implementations of MCD-based BayesNNs in [23]. Although there are some other BayesNN accelerators [19], [27], they do not report any end-to-end latency and energy consumption.

As shown in Table II, our design achieves 65 and 33 times higher energy efficiency than CPU and GPU implementations, despite the FPGA adopting 20nm technology while the CPU adopting 14nm technology and the GPU adopting 12nm technology. Our accelerator also shows lower latency and better energy efficiency than both [22] and [20]. Although both [26] and [23] are faster than our design, they consume much higher energy due to the high resource utilization and frequent data transfer between on-chip and off-chip memory, leading to nearly 5 and 4 times lower energy efficiency than our design. Also, compared with their Verilog-based implementations, our HLS-based accelerator has advantages in development time [28], which can improve designer productivity and can facilitate extending our approach to cover other NNs such as LSTM [13]. Table III provides the power consumption breakdown obtained from the Xilinx Power Estimator (XPE) tool after place and route. The dynamic power occupies 72% of the total power. The logic&signal and IO consume most of the dynamic power, accounting for 30% and 21%, respectively. The high IO power consumption results from our spatial mapping strategy with multiple MC engines running in parallel.

VI. CONCLUSION

This paper proposes a novel multi-exit Monte-Carlo Dropout (MCD)-based Bayesian Neural Networks (BayesNNs). To facilitate its deployment in real-life applications, a transformation framework is developed to produce FPGA-based accelerators for multi-exit MCD-based BayesNNs. Several novel hardware optimizations are introduced for performance improvement. Comprehensive experiments demonstrate that our approach achieves higher algorithmic and energy efficiency than state-of-the-art designs. In the future, we aim to optimize the design with zero skipping, support attention-based BayesNNs, and include capabilities such as run-time reconfiguration.

ACKNOWLEDGEMENT

The support of UK EPSRC grants (UK EPSRC grants EP/L016796/1, EP/N031768/1, EP/P010040/1, EP/V028251/1 and EP/S030069/1) is gratefully acknowledged.

REFERENCES

[1] S. Dong *et al.*, "A survey on deep learning and its applications," *Computer Science Review*, vol. 40, p. 100379, 2021.
[2] R. M. Neal, "Bayesian learning via stochastic dynamics," in *Advances in Neural Information Processing Systems (NeurIPS)*, 1993, pp. 475–482.
[3] C. Blundell *et al.*, "Weight uncertainty in neural network," in *International Conference on Machine Learning (ICML)*, 2015, pp. 1613–1622.
[4] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *International Conference on Machine Learning (ICML)*, 2016, pp. 1050–1059.
[5] Y. Ovadia *et al.*, "Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.
[6] J. Fowers *et al.*, "A configurable cloud-scale dnn processor for real-time ai," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 1–14.
[7] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.
[8] H. Fan *et al.*, "Adaptable butterfly accelerator for attention-based NNs via hardware and algorithm co-design," in *MICRO-55: 55th Annual IEEE/ACM International Symposium on Microarchitecture*, 2022.
[9] C. Zhang *et al.*, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2072–2085, 2018.
[10] F. Fahim *et al.*, "hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices," *arXiv preprint arXiv:2103.05579*, 2021.
[11] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.
[12] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[14] S. Fort *et al.*, "Deep ensembles: A loss landscape perspective," *arXiv preprint arXiv:1912.02757*, 2019.
[15] L. Qendro *et al.*, "Early exit ensembles for uncertainty quantification," in *Proceedings of Machine Learning for Health*, ser. Proceedings of Machine Learning Research, vol. 158. PMLR, 2021, pp. 181–195.
[16] G. Huang *et al.*, "Multi-scale dense networks for resource efficient image classification," in *International Conference on Learning Representations (ICLR)*, 2018.
[17] H. Lee and J.-S. Lee, "Students are the best teacher: Exit-ensemble distillation with multi-exits," *arXiv preprint arXiv:2104.00299*, 2021.
[18] Y. Kaya *et al.*, "Shallow-deep networks: Understanding and mitigating network overthinking," in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97. PMLR, 2019, pp. 3301–3310.
[19] Q. Wan *et al.*, "Shift-BNN: Highly-efficient probabilistic bayesian neural network training via memory-friendly pattern retrieving," in *2021 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021, pp. 885–897.
[20] H. Awano and M. Hashimoto, "BYNQNET: Bayesian neural network with quadratic activations for sampling-free uncertainty estimation on FPGA," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1402–1407.
[21] H. Fan *et al.*, "FPGA-based acceleration for bayesian convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 12, pp. 5343–5356.
[22] R. Cai *et al.*, "VIBNN: Hardware acceleration of bayesian neural networks," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 476–488, 2018.
[23] H. Fan *et al.*, "Accelerating Bayesian neural networks via algorithmic and hardware optimizations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3387–3399, 2022.
[24] A. Kendall *et al.*, "Bayesian Segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," *arXiv preprint arXiv:1511.02680*, 2015.
[25] L. Qendro *et al.*, "Stochastic-Shield: A probabilistic approach towards training-free adversarial defense in quantized cnns," in *Proceedings of the 1st Workshop on Security and Privacy for Mobile AI*, 2021, p. 1–6.
[26] H. Fan *et al.*, "High-performance FPGA-based accelerator for Bayesian neural networks," in *Proceedings of the 2021 ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1–6.
[27] Q. Wan *et al.*, "Fast-BCNN: Massive neuron skipping in Bayesian convolutional neural networks," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 229–240.
[28] M. Pelcat *et al.*, "Design productivity of a high level synthesis compiler versus HDL," in *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. IEEE, 2016, pp. 140–147.