Algorithm and Hardware Co-Design for Multi-Exit Dropout-based Bayesian Neural Networks

Hongxiang Fan†, Hao Chen†, Liam Castelli, Martin Ferianc, Wayne Luk, *Fellow, IEEE*

***Abstract*—Reliable uncertainty prediction plays a crucial role in various safety-critic applications such as medical diagnosis and autonomous driving. In recent years, Bayesian Neural Networks (BayesNNs) have gained substantial research and industrial interests due to their remarkable capability to estimate reliable uncertainty. Nevertheless, the algorithmic complexity and hardware performance of BayesNNs for uncertainty estimation hinder their adaptation in real-life scenarios. To bridge this gap, this paper proposes an algorithm and hardware co-design framework that can generate FPGA-based accelerators for efficient uncertainty prediction. At the algorithmic level, we propose novel multi-exit mask-based BayesNNs, which effectively decreases the computational and memory overhead while achieving high-quality uncertainty estimation. At the hardware level, this paper introduces a transformation framework that can generate FPGA-based accelerators for the proposed efficient algorithms. Several hardware optimization techniques such as partial masking and the mix of spatial and temporal mappings are introduced to reduce resource consumption and improve the overall hardware performance. Comprehensive experiments demonstrate that our approach can achieve higher energy efficiency compared to CPU, GPU, and other state-of-the-art hardware implementations. To support the future development of this research, we have open-sourced our code at: https://github.com/os-hxfan/MCME_FPGA_Acc.git**

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

I. Introduction

Deep neural networks (DNNs) have emerged as a cutting-edge frontier of artificial intelligence, with extensive applications in various domains ranging from computer vision [1] to natural language processing [2]. However, conventional DNNs are suffering from a critical limitation: they operate akin to black boxes, rendering them incapable of explaining their predictive results or providing an estimate of the uncertainty in their correctness [3]. The lack of reliable uncertainty estimation undermines the trustworthiness of conventional DNNs, making them unsuitable candidates for safety-critic applications [4], [5], [6] where reliable confidence and uncertainty measures are imperative.

This work was supported in part by the United Kingdom EPSRC under Grant EP/L016796/1, Grant EP/N031768/1, Grant EP/P010040/1, Grant EP/V028251/1 and Grant EP/S030069/1, Maxeler, Intel, Xilinx and SGIIT.

H. Fan is with Samsung AI Center, Cambridge, CB1 2JH, UK. He is also affiliated with the Department of Computer Science and Technology, University of Cambridge, CB3 0FD, UK.

H. Chen, L. Castelli, Z. Zhang and W. Luk are with the Department of Computing, Imperial College London, London, SW7 2AZ, UK.

M. Ferianc is with the Department of Electronic and Electrical Engineering, University College London, London, WC1E 6BT, UK.

† Equal Contribution.

* Corresponding author: Hongxiang Fan (h.fan17@imperial.ac.uk).

Bayesian Neural Networks (BayesNNs) [7] leverage Bayesian inference to model the predictive uncertainty, which addresses the limitation of conventional DNNs in estimating uncertainty. By representing the weights as probabilistic distributions, BayesNNs provide a principled approach to quantifying uncertainty, enhancing the robustness and trustworthiness of their predictions. Nevertheless, the benefits of BayesNNs also come with costs: the high dimensionality of modern BayesNNs introduces prohibitively expensive computational and memory overheads, making the exact Bayesian inference intractable [8].

Although various approximation approaches, such as Bayes-by-backprop [9] and Monte-Carlo Dropout (MCD) [10], have been introduced to reduce the algorithmic complexity of BayesNNs, there are still two challenges while deploying these approximated BayesNNs in real scenarios. First, these approximation methods generally perform worse than traditional deep ensembles with respect to the performance of uncertainty estimation [11]. 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 [12], [13], [14] or design automation tools [15], [16] for non-BayesNNs such as convolutional NNs (CNNs) [17], [18] and long short-term memory (LSTM) [19]. Hence there is an urgent need for publicly accessible hardware acceleration for BayesNNs.

To reduce the barrier of deploying BayesNNs in real applications, this paper proposes an algorithm and hardware co-design approach to improve the performance of BayesNNs. At the algorithm level, we propose a novel multi-exit MCD-based BayesNN that attains low computational and memory overheads while achieving better uncertainty estimation than traditional deep ensembles. We further extend the multi-exit concept on Masksemble [20], an efficient variant of MCD-based BayesNN without the need for runtime sampling. Both approaches fall under the category of multi-exit dropout-based BayesNNs with each demonstrating unique trade-offs between algorithmic and hardware performance. At the hardware level, we choose FPGA as our acceleration platform due to its superior flexibility over Application-Specific Integrated Circuit (ASIC) and its potential for achieving higher energy efficiency over Graphics Processing Units (GPUs) [21]. As shown in Figure 1, we propose a transformation framework to generate high-performance FPGA-based accelerators of multi-exit dropout-based BayesNNs for efficient uncertainty

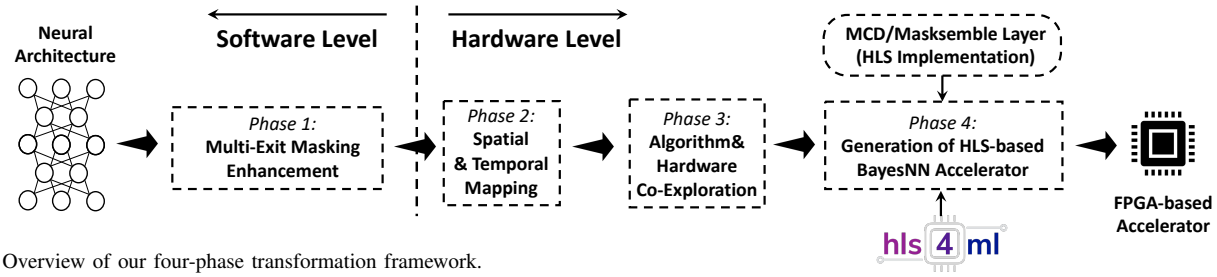


Fig. 1. Overview of our four-phase transformation framework.

estimation. 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. To facilitate public access to our artifacts, we open-source our code at https://github.com/os-hxfan/MCME_FPGA_Acc.git.

The contributions of this paper can be summarized as follows:

- Novel multi-exit dropout-based Bayesian Neural Network (BayesNNs) approaches that achieve high quality of uncertainty estimation with low compute and memory overheads.
- A design framework for generating FPGA-based accelerators for multi-exit dropout-based BayesNNs with high hardware performance and energy efficiency.
- Various optimization strategies including partial dropout and spatial-temporal mapping and algorithm-hardware co-exploration for performance improvement.
- A comprehensive evaluation of the proposed approach based on multiple models and datasets, demonstrating the effectiveness of our co-design approach in improving both algorithm and hardware performance.

This work extends our conference publication [3]. The extended material includes: *1)* multi-exit support on Masksembles to improve their algorithmic performance; *2)* FPGA-based acceleration of multi-exit Masksemble with optimized implementation to improve hardware performance; *3)* a more comprehensive evaluation on the quality of uncertainty estimation across multiple models and datasets.

II. Background and Related Work

A. Bayesian Neural Networks

Bayesian Neural Networks (BayesNNs) demonstrate the capability to effectively mitigate overfitting and enable the estimation of the model uncertainty through the utilization of Bayesian inference. In contrast to the non-BayesNNs which only capture point-wise weight values, BayesNNs are able to acquire knowledge about the distribution of the weights. The training process involves the use of Bayes rule to determine the distribution $p(w|D)$ where w is the weights corresponding to the training data D. Despite their advantages, the current BayesNNs have limited utility in real-world settings because their high dimensionality renders the analytical calculation of the posterior distribution $p(w|D)$ computationally infeasible.

There are two main streaming approaches aiming to approximate the intractable Bayesian inference required by BayesNNs: Markov chain Monte Carlo (MCMC) and variational inference (VI) [22]. MCMC-based methods directly sample from exact posterior distributions, and representative algorithms include Hamiltonian Monte Carlo (HMC) [23] and stochastic gradient Langevin Dynamic (SGLD) [24] approaches. Instead of sampling from the exact posterior, VI-based approaches deploy variational distributions with a set of variational parameters θ. During training, θ is optimized to ensure the variational parameters are as close as possible to the exact posterior.

B. Dropout-based Approximations for BayesNNs

1) MCD-based BayesNNs: Monte-Carlo dropout (MCD) can be categorized as one of VI-based approximation approaches that adopt dropout masks to perform efficient uncertainty estimation [22]. MCD implements a random filter-wise binary mask to randomly removes connections between layers of a neural network. The mask values M_i follow a Bernoulli distribution $p(M_i|p_i)$, where the binary random variables take on the value of 0 with a drop rate p_i. It has been proven that MCD could be interpreted mathematically as approximate Bayesian inference for deep Gaussian processes [10]. A key distinction between dropouts traditionally employed in non-BayesNNs and MCD is that MCD use dropouts in both training and inference. During inference, MCD-based BayesNNs execute multiple forward passes and the results are obtained by averaging the output of the generated MC samples. Each forward pass uses an independently generated set of masks, allowing for the calculation of uncertainty and calibration.

2) Masksemble: By leveraging the predictive power of multiple independent DNNs, deep ensembles [25] can significantly increase accuracy and the quality of uncertainty [25]. They are reported to perform better than various approximated BayesNNs across different algorithmic metrics while achieving higher robustness against dataset shift [11]. Whereas, deep ensembles require the user to train and maintain multiple DNNs in parallel, remarkably increasing the computational and memory cost.

Inspired by MCD-based BayesNNs, Masksembles [20] apply multiple sets of pre-defined dropout masks on one shared single DNN, effectively reducing the memory overhead. Besides, there are another two advantages of Masksembles when compared to MCD-based BaeysNN First, since the dropout masks are determined before training and inference, Masksembles eliminate the need for runtime sampling, which effectively reduces the hardware cost. Second, the overlap and correlation among different dropout masks in Masksembles can be strictly controlled, making it achieve a similar algorithmic performance as traditional deep ensembles.

C. Multi-Exit DNNs

Conventional deep learning architectures typically employ a single exit per network to generate final predictive outputs. However, this single-exit architecture exhibits two drawbacks when processing inputs that necessitate only intermediate features extracted from the middle layers. First, unnecessary computation and memory costs incur as single-exit DNNs always process all the layers until the final exit even when the intermediate features are informative enough for predictions. Second, certain key features extracted from the intermediate layers might get lost as the network goes deeper, resulting in inaccurate prediction. To avoid these issues, Multi-exit [26] DNNs are introduced that outputs results at various points in a single pass to improve the performance and avoid redundant processing.

While some architectures are specially designed to function with these additional exits, like the Multi-Scale DenseNet, best performance is usually obtained through attaching multiple classifiers to high-performance networks like ResNet [27]. Common choices for where to attach the classifiers is after specific amounts of floating-point operations (FLOPs) or groupings of convolutional layers [28], [29]. In this paper, we adopt the multi-exit enhancement as an approach to improve the accuracy, uncertainty estimation and compute efficiency of BayesNNs.

D. Related Work

Extensive research has been conducted on DNNs and the use of FPGAs to accelerate them for various applications [1]. Representative work includes energy-efficient CNN acceleration [13] and FPGA-based real-time AI cloud services [12]. Significant research also targets design automation for DNNs, like the open source tool *hls4ml* supporting an automatic design flow involving high-level synthesis to promote low-power machine learning [16].

FPGA-based acceleration of BayesNNs has emerged recently [30]. Early designs include *Bynqnet*, an FPGA-based BayesNNs with quadratic activations for sampling-free uncertainty estimation [31]. Efficient FPGA implementations for 2D and 3D convolutional BayesNNs have been proposed in [32]. For recurrent Bayesian DNNs, [33] proposed an FPGA accelerator as well as an algorithmic co-design framework. Another work is *VIBNN*, an FPGA-based accelerator that supports Gaussian distribution-based BaynesNNs sampled at runtime [34]. Additionally, [8] proposed algorithmic and hardware optimizations for BayesNNs, exploiting their structured sparsity and redundant computations. Lastly, [35] explored quantisation in BayesNNs enabling their efficient execution on FPGAs using integer arithmetic.

In contrast to these approaches, this work extends and differs from the related work in several ways. First, it proposes a novel multi-exit dropout-based Bayesian DNN, which effectively decreases the computational and memory overhead while achieving high-quality uncertainty estimation and accuracy. Second, it introduces an automatic pipeline which translates a software description of the multi-exit BayesNN into a hardware design, executable on an FPGA. Third, it introduces several optimization techniques to reduce overall resource consumption and improve the hardware performance of multi-exit BayesNNs without harming their algorithmic performance. These contributions are generalisable to different datasets and DNN architectures, as shown in the experiments, and extensible to previous work mainly through the addition of sampling-based early exits and their hardware consideration.

III. Multi-Exit Masked BayesNNs

A. Multi-Exit Enhancement

As mentioned in Section II-B, while both MCD-based BayesNNs and Masksembles demonstrate the potential for efficient uncertainty estimation, they still suffer clear limitations. On one side, MCD-based approximation methods have been subject to criticism due to the inferior performance in uncertainty estimation and calibration ability when compared to deep ensembles [11]. It has been empirically shown that the introduction of MCD layers after activations in vanilla MCD-based BayesNNs can hamper their predictive power, worsening both its accuracy and its uncertainty quantification [36]. On the other side, Masksembles impose a heavy computational burden since obtaining each output sample necessitates running the entire network. This compute inefficiency hinders their widespread adoption for efficient uncertainty estimation. To address these drawbacks, this paper proposes a novel multi-exit enhancement approach for both MCD-based BayesNNs and Masksembles. By adopting this approach, we aim to achieve effective and efficient uncertainty prediction, mitigating the limitations of each method.

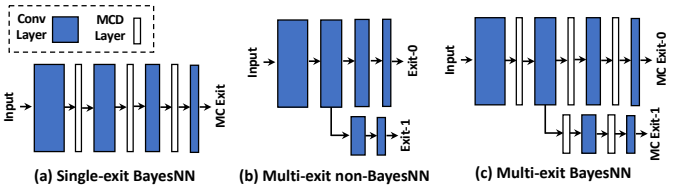


Fig. 2. Difference between a single-exit BayesNN, a multi-exit NN and a multi-ext BayesNN.

1) Multi-Exit MCD-based BayesNNs: Figure 2 presents the network architectures of three distinct approaches: a vanilla MCD-based BayesNN, a multi-exit non-BayesNN, and a multi-exit MCD-based BayesNN. By applying multi-exit enhancement on vanilla MCD-based BayesNNs, we propose multi-exit MCD-based BayesNNs, as decipted in Figure 2(c). In contrast to the traditional single-exit MCD-based BayesNNs, our multi-exit MCD-based approach excels in incorporating outputs from different exits, effectively improve the quality of uncertainty estimation. Furthermore, when compared to multi-exit non-BayesNNs, our proposed approach has the advantage of generating arbitrary numbers of MC samples with the use of MCD layers, improving the flexibility for uncertainty estimation. An intriguing aspect of multi-exit MCD-based BayesNNs lies in the capability of capturing the uncertainty information across different network stages. This stems from the utilization of diverse intermediate features extracted from different stages of the network to generate predictive results from multiple exits.

2) Multi-Exit Masksembles: Although the use of MCD layers enables flexibility in generating arbitrary MC samples, it also introduces hardware overhead due to the frequent runtime Bernoulli sampling. To provide a hardware-efficient alternative, we propose multi-exit Masksembles that replace MCD layers with Masksemble layers. To avoid the highly correlated predictive results across multiple exits, we adopt the mask scale parameter [20] to control the overlap among different pre-defined masks. There are two distinct computational differences when comparing MCD- and Masksemble-based approaches. First, by adopting pre-defined binary masks, multi-exit Masksembles eliminate the need for runtime sampling. As the locations of zeros are fixed, it provides us the opportunity for designing efficient hardware accelerators to intelligently skip redundant computation associated with zero values Section IV-E. Second, MCD-based method applies dropout in the channel granularity, while Masksemble layer adopts point-wise masks with more fine-grained dropout granularity. These two difference leads to distinct hardware design requirements while accelerating multi-exit MCD-based BayesNNs and multi-exit Masksembles.

In this paper, we treat both MCD and Maskensemble layers as two distinct dropout layers, each exhibiting specific trade-offs among accuracy, uncertainty and hardware performance. To fulfil the diverse needs of different users, we propose a co-design framework dedicated to optimizing dropout layers, as elaborated in Section IV. This optimization enables users to tailor dropout layers for their target applications, ultimately leading to efficient uncertainty estimation for various scenarios.

B. Partial Dropout

Applying dropout layers after every convolution incurs large computational overhead since it requires running the whole network multiple times to get different MC samples. Inspired by [11], [37], [38], [3], we propose partial dropout for both multi-ext Masksembles and multi-exit MCD-based BayesNNs. Rather than adopting the fully dropout approach, we insert dropout layers starting from exits towards the input. We refer to the layers without dropout applied as the non-dropout component. By placing dropout layers closer to each exit, fewer computations are required since the non-dropout results can be cached and reused for different MC samples.

With partial dropout applied, both multi-exit MCD-based BayesNN and multi-exit Masksemble can be interpreted as ensembles of approximated BayesNNs built upon the feature space. To understand this, one can interpret the non-dropout components as feature extractors. 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 $1 \leq i \leq M$. Build upon the features extracted by $f_i(X)$, each exit then adopts the dropout-based Bayesian approach through either MCD or Masksemble layer to perform uncertainty estimation. The final result ensembles predictions from different approximated BayesNNs in multiple exits. Therefore, our approach can be interpreted as ensembles of dropout-based BayesNNs built upon the feature space.

C. Compute Efficiency

We demonstrate that our proposed multi-exit dropout-based BayesNNs have higher compute efficiency over single-exit BayesNNs for generating MC samples. 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:

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

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

$$FLOP_{main} + \frac{N_{sample}}{N_{exit}} \times FLOP_{exit}. \tag{2}$$

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

$$\frac{1+\alpha}{\frac{1}{N_{sample}} + \frac{\alpha}{N_{exit}}}, \tag{3}$$

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-C 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 [28]. 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 [28] is used.

IV. Transformation Framework

A. Framework Overview

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

Given the neural architecture description as an input, the first phase extends it with multi-exit masking with either MCD [10] or Masksemble [20] according to the user requirements. 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 strategies. The last phase generates the corresponding HLS-based hardware accelerator. We adopt the design flow and HLS template of common DNN layers from *HLS4ML* [16]. To support the generation of accelerators for reliable uncertainty prediction, we add the HLS-based implementation of MCD/Masksemble layers and *Keras*-to-HLS conversion into the design flow.

B. Multi-Exit Masking: Phase 1

Multi-exit masking phase aims to optimize different design parameters of *MEMBsemble*, including the number of exits N_{exit}, the number of forward passes N_{pass}, the type of masking layers and the associated masking parameters. These design parameters decide the trade-off between accuracy, uncertainty estimation and hardware performance. For instance, the total number of MC samples N_{sample} obtained from a *MEMBsemble* with N_{exit} exits and N_{pass} forward passes is given by $N_{sample} = N_{pass} \times N_{exit}$. The higher number of N_{exit} and N_{pass} may improve both the accuracy and calibration. However, higher N_{sample} can degrade hardware performance due to the larger computational and memory demands. As different applications and tasks may have different requirements for algorithmic 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 with either MCD or Masksemble layers, different *MEMBsemble* candidates are constructed and trained on the target dataset. When the training finishes, we evaluate different metrics for all candidates, including accuracy, calibration and the number 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 number of FLOPs. The final optimized design is fed into the next stage for hardware design generation.

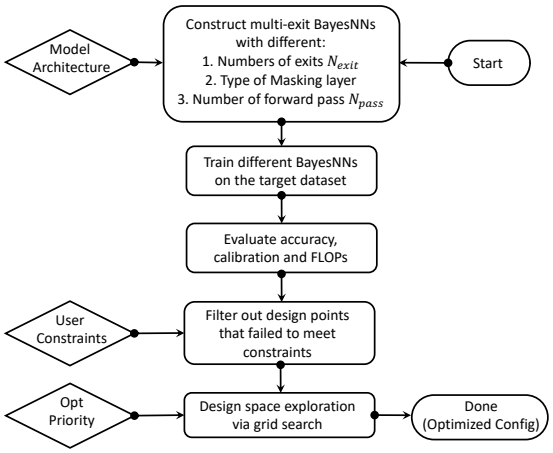


Fig. 3. Optimization flow.

C. Spatial and Temporal Mappings: Phase 2

The inference of masking components requires multiple forward passes to obtain different MC samples. This masking-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 *MEMBsemble*, which are illustrated in Figure 4.

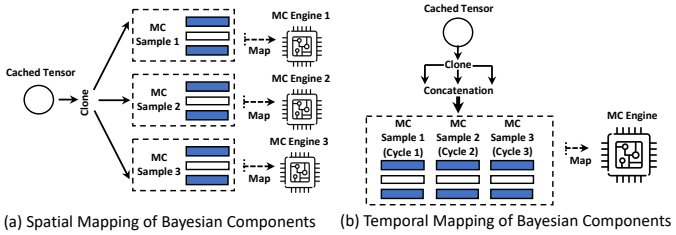


Fig. 4. Spatial and temporal mappings for masking components.

In both mapping strategies, the tensor generated from the last non-masking 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 computational resource use when the number of MC samples becomes high. 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 algorithmic 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 spaces.

E. Generation of FPGA-based Accelerator: Phase 4

The generation of hardware accelerators is based on *HLS4ML* and our customized design templates for MCD and Masksemble layers. The generated HLS-based accelerators can then be fed into Vivado-HLS for synthesis and implementation to get the final bitstream for onboard testing. The pseudocodes of HLS-based implementation of MCD and Masksemble layers are presented in Algorithm 1 and 2, respectively. In both implementations, 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. In the HLS-based MCD 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*. In the HLS implementation of the Masksemble layer, the dropout masks are fed as inputs into the hardware module, eliminating the hardware cost of Bernoulli sampling. The inputs with mask values being one are passed through to the outputs.

Algorithm 1 Pseudocode of MCD layer

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

Algorithm 2 Pseudocode of Masksemble layer

```
Input: input[mask_size], mask_index,
       generated_masks[mask_num][mask_size]
Output: output[mask_size]
for (i from 0 to mask_size) do    ▷ #pragma PIPELINE
    mask_value = generated_masks[mask_index][i]
    if (mask_value == 0) then
        output[i] = 0
    else
        output[i] = input[i]
```

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. Resource Cost of Being Bayesian

Inserting dropout layers transforms conventional DNNs into BayesNNs, enabling reliable uncertainty estimation required by various safety-critic applications. To quantitatively investigate the hardware overhead imposed by this Bayesian transformation, we evaluate the resource consumption of Bayesian accelerators against their non-Bayesian counterparts. Three BayesNNs and datasets are used in our experiments, i.e., *LetNet5* on MNIST, *ResNet-18* on CIFAR-10, and *VGG-11* on SVHN. As we aim to evaluate the resource cost of being Bayesian, all the models use single-exit to eliminate the hardware overhead introduced by the multi-exit optimization. We generate different Bayesian accelerators with distinct numbers of dropout layers using our proposed design flow Section IV. For non-Bayesian accelerators, we set the number of dropout layers as zero. In order to fit BayesNNs onto FPGA, we apply quantization and custom channel numbers to ease the memory requirements. To further reduce compute resource consumption, we adopt temporal mapping on all the hardware designs.

Figure 6 shows the resource consumption of Block RAM (BRAM), DSP, Flip-Flop (FF) and LUT. We implement two different dropout layers, MCD and Masksemble, for each model with varied numbers of dropout layers. As can be observed in all three models, the BRAM and DSP usage stays almost the same across different numbers of dropout layers and dropout types. The reason is that dropout layers do not contain compute- and memory-intensive operations. The designs of both MCD and Masksemble layers can be implemented by mainly just using logic resources. In contrast, an increasing trend can be observed in both FF and LUT consumption when more dropout layers are inserted. The most significant increase is found on MCD-based Bayes-VGG, where nearly 13% more FF resources are utilized for the insertion of 8 dropout layers. However, this overhead is caused by inserting MCD layers after every convolution. With our proposed partial dropout Section III-B, the LUT and FF resource overheads of one MCD layer are just around $1\% \sim 2\%$, demonstrating the resource-efficiency of our co-design approach.

B. Improvement of Hardware Optimizations

To demonstrate the improvement introduced by different hardware optimizations, we evaluate the latency performance of different BayesNN accelerators generated by our transformation framework.

Masksemble against MCD

Spatial mapping against temporal mapping.

C. Effect of Multi-Exit Enhencement

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.

As discussed previously, the usage of too many dropout layers in a BayesNN can overregularize the network and

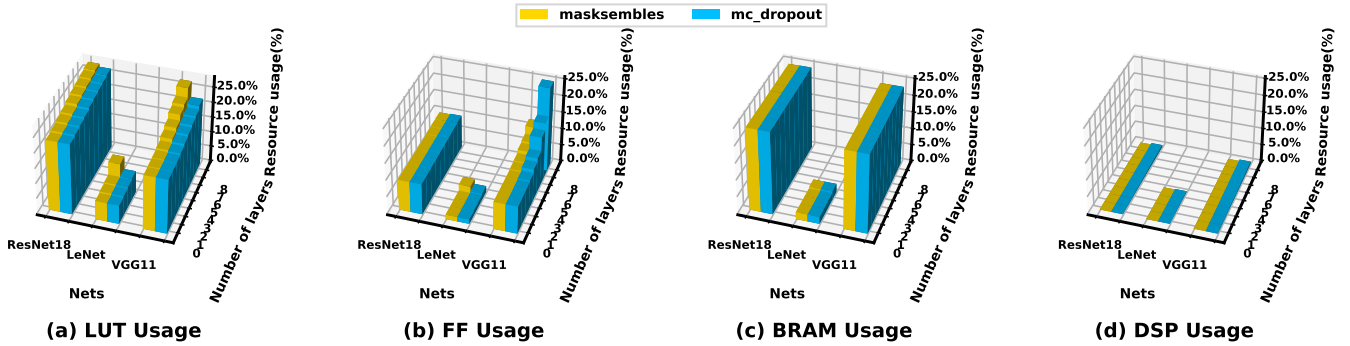


Fig. 5. Resource consumption of mask-based BayesNNs with LeNet, ResNet18 and Bayes-VGG11 as network backbones. The quantization and custom number of channels are applied to fit models onto FPGAs.

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
Mask	$XXX \pm XXX$	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	$\mathbf{0.017 \pm 0.001}$	$\mathbf{0.45 \pm 0.02}$
Mask+ME (Ours)	$\boldsymbol{xx \pm xx}$	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 [34]**	**DATE'20 [31]**	**DAC'21 [3]**	**TPDS'22 [8]**	**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

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 [29]: 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 [26].

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) [8]: 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.

D. 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 [34], [31], [3], [8]. As both [34] and [31] 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 [8]. Although there are some other BayesNN accelerators [39], [30], 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 [34] and [31]. Although both [3] and [8] 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 [40], which can improve designer productivity and can facilitate extending our approach to cover other NNs such as LSTM [19]. 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.

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%

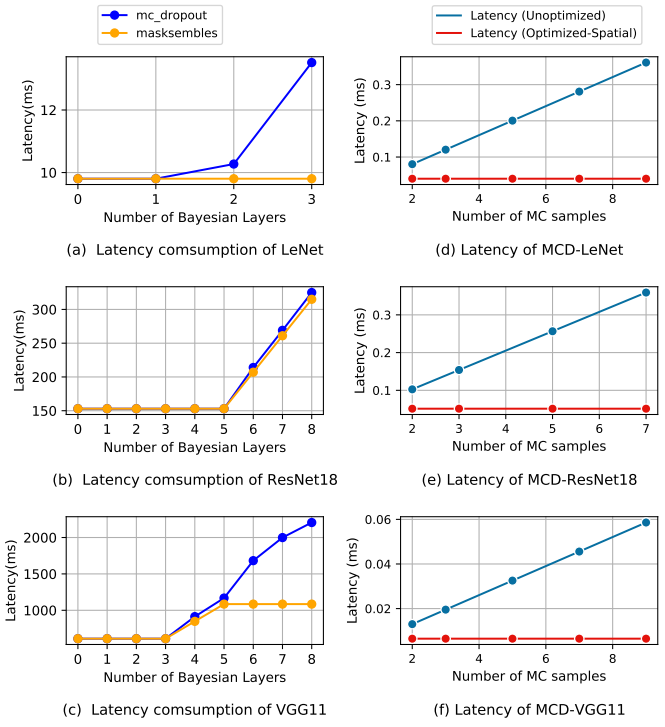


Fig. 6. Resource consumption and latency of Bayes-LeNet, Bayes-ResNet18 and Bayes-VGG11 with quantization and custom number of channels.

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.

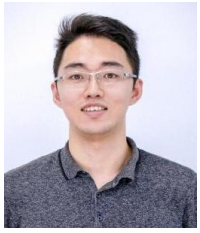
ACKNOWLEDGEMENT

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] D. W. Otter *et al.*, "A survey of the usages of deep learning for natural language processing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 2, pp. 604–624, 2020.
[3] 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.
[4] C. Leibig *et al.*, "Leveraging uncertainty information from deep neural networks for disease detection," *Scientific Reports*, vol. 7, no. 1, pp. 1–14, 2017.
[5] F. Liang, Q. Li, and L. Zhou, "Bayesian neural networks for selection of drug sensitive genes," *Journal of the American Statistical Association*, vol. 113, no. 523, pp. 955–972, 2018.
[6] T. Azevedo *et al.*, "Stochastic-yolo: Efficient probabilistic object detection under dataset shifts," 2020.
[7] R. M. Neal, "Bayesian learning via stochastic dynamics," in *Advances in Neural Information Processing Systems (NeurIPS)*, 1993, pp. 475–482.
[8] 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.
[9] J. M. Hernández-Lobato and R. Adams, "Probabilistic backpropagation for scalable learning of bayesian neural networks," in *International Conference on Machine Learning*. PMLR, 2015, pp. 1861–1869.

[10] 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.
[11] 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.
[12] 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.
[13] 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.
[14] 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.
[15] 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.
[16] F. Fahim *et al.*, "hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices," *arXiv preprint arXiv:2103.05579*, 2021.
[17] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.
[18] 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.
[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[20] N. Durasov *et al.*, "Masksembles for uncertainty estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 13 539–13 548.
[21] Y. Ma *et al.*, "Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 45–54.
[22] L. V. Jospin *et al.*, "Hands-on bayesian neural networks—a tutorial for deep learning users," *IEEE Computational Intelligence Magazine*, vol. 17, no. 2, pp. 29–48, 2022.
[23] R. M. Neal *et al.*, "Mcmc using hamiltonian dynamics," *Handbook of Markov Chain Monte Carlo*, vol. 2, no. 11, p. 2, 2011.
[24] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," in *International Conference on Machine Learning (ICML)*, 2011, pp. 681–688.
[25] S. Fort *et al.*, "Deep ensembles: A loss landscape perspective," *arXiv preprint arXiv:1912.02757*, 2019.
[26] 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.
[27] G. Huang *et al.*, "Multi-scale dense networks for resource efficient image classification," in *International Conference on Learning Representations (ICLR)*, 2018.
[28] H. Lee and J.-S. Lee, "Students are the best teacher: Exit-ensemble distillation with multi-exits," *arXiv preprint arXiv:2104.00299*, 2021.
[29] 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.
[30] 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.
[31] 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.
[32] 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, 2022.
[33] M. Ferianc, Z. Que, H. Fan, W. Luk, and M. Rodrigues, "Optimizing bayesian recurrent neural networks on an fpga-based accelerator," in *2021 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2021, pp. 1–10.
[34] R. Cai *et al.*, "VIBNN: Hardware acceleration of bayesian neural networks," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 476–488, 2018.
[35] M. Ferianc, P. Maji, M. Mattina, and M. Rodrigues, "On the effects of quantisation on model uncertainty in bayesian neural networks," in *Uncertainty in Artificial Intelligence*. PMLR, 2021, pp. 929–938.
[36] A. Kendall *et al.*, "Bayesian Segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," *arXiv preprint arXiv:1511.02680*, 2015.
[37] A. Kristiadi *et al.*, "Being bayesian, even just a bit, fixes overconfidence in relu networks," *arXiv preprint arXiv:2002.10118*, 2020.
[38] A. Kendall *et al.*, "Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," *arXiv preprint arXiv:1511.02680*, 2015.
[39] 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.
[40] 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.



Hongxiang Fan received the B.S. degree in electronic engineering from Tianjin University, Tianjin, China, in 2017, and the master's degree from the Department of Computing, Imperial College London, London, U.K., in 2018. He is currently a Ph.D. student in Machine Learning and High-Performance Computing at Imperial College London. His current research focuses on efficient algorithm and acceleration for Machine Learning applications.

Hao (Mark) Chen is a final-year MEng student at Imperial College London. His reseaerch interests include machine learning systems, domain-specific languages for embedded systems, and software-hardware co-design.

Martin Ferianc obtained an MEng in Electronic and Information Engineering from Imperial College London, London, UK in 2015. He is currently a PhD candidate in the Department of Electronic and Electrical Engineering at University College London. His research interests include Bayesian neural networks, deep learning and hardware acceleration of neural networks.

Wayne Luk (Fellow, IEEE) received the M.A., M.Sc., and D.Phil. degrees in engineering and computing science from Oxford University, Oxford, U.K. He founded and leads the Custom Computing Group, Department of Computing at Imperial College London, where he is Professor of Computer Engineering. He was a Visiting Professor at Stanford University, Stanford, CA, USA. Dr. Luk is a Fellow of the Royal Academy of Engineering and the BCS. He had 15 papers that received awards from international conferences, and he received a Research Excellence Award from Imperial College London. He was a founding Editor-in-Chief of the ACM Transactions on Reconfigurable Technology and Systems, and has been a member of the Steering Committee and Program Committee of various international conferences.