Accelerating Bayesian Neural Networks via Algorithmic and Hardware Optimizations

Hongxiang Fan*, Martin Ferianc, Zhiqiang Que, Xinyu Niu, Miguel Rodrigues, *Senior Member, IEEE*, Wayne Luk, *Fellow, IEEE*

Abstract—Bayesian neural networks (BayesNNs) have demonstrated their advantages in various safety-critical applications, such as autonomous driving or healthcare, due to their ability to capture and represent model uncertainty. However, standard BayesNNs require to be repeatedly run because of Monte Carlo sampling to quantify their uncertainty, which puts a burden on their real-world hardware performance. To address this performance issue, this paper systematically exploits the extensive structured sparsity and redundant computation in BayesNNs. Different from the unstructured or structured sparsity in standard convolutional NNs, the structured sparsity of BayesNNs is introduced by Monte Carlo Dropout and its associated sampling required during uncertainty estimation and prediction, which can be exploited through both algorithmic and hardware optimizations. We first classify the observed sparsity patterns into three categories: channel sparsity, layer sparsity and sample sparsity. On the algorithmic side, a framework is proposed to automatically explore these three sparsity categories without sacrificing algorithmic performance. We demonstrated that structured sparsity can be exploited to accelerate CPU designs by up to 49 times, and GPU designs by up to 40 times. On the hardware side, a novel hardware architecture is proposed to accelerate BayesNNs, which achieves a high hardware performance using the runtime adaptable hardware engines and the intelligent skipping support. Upon implementing the proposed hardware design on an FPGA, our experiments demonstrated that the algorithm-optimized BayesNNs can achieve up to 56 times speedup when compared with unoptimized Bayesian nets. Comparing with the optimized GPU implementation, our FPGA design achieved up to 7.6 times speedup and up to 39.3 times higher energy efficiency.

Index Terms—Bayesian neural network (BayesNN), Structured sparsity, Field-programmable gate array (FPGA), Deep learning

1 Introduction

Neural networks (NNs) have become one of the most effective algorithms in computer vision. They have been widely deployed in various artificial intelligence (AI) applications such as in object detection [1] or scene segmentation [2]. However, standard NNs are incapable of quantifying their uncertainty [3], so they are unsuitable for safety-critical applications such as those in autonomous driving, medicine or chemistry [1], [4], [5], [6], [7]. For instance, physicians or self-driving systems can be deceived by a standard NN which does not quantify the level of uncertainty in its output.

As a variant of NNs, a Bayesian NN (BayesNN) [4], [8], [9] has become an appealing solution for safety-critical applications since it can quantify the uncertainty of its output. Gal *et al.* [4] demonstrated a BayesNN can be obtained by applying the Monte Carlo Dropout (MCD) after every convolutional layer. Figure 1 shows a comparison between a BayesNN and a standard NN for image classification, with confidence reflected by the predictive probability of class labels. While feeding a previously seen input image into the networks, both networks make the correct prediction and their confidence is justifiable. However, given random noise input, the standard NN is completely overconfident and wrong, while the BayesNN can make use of its uncertainty estimation capability to lower its confidence. Hence BayesNNs, along with their robustness to overfitting [10], have become popular in applications where uncertainty quantification is essential [1], [5], [6], [7], [11].

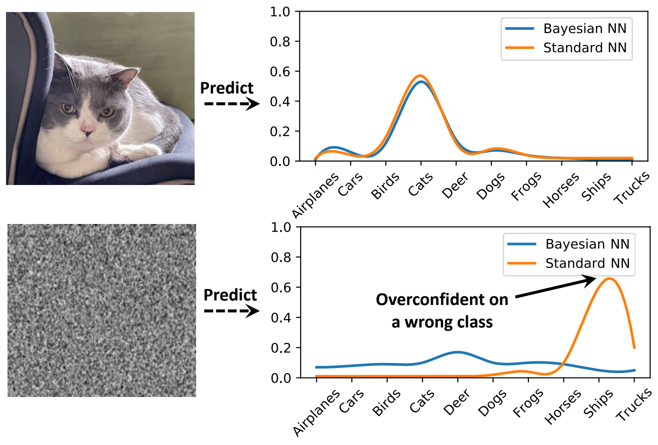


Fig. 1. Standard NN is overconfident on random inputs, when compared with a BayesNN.

Nevertheless, the proper uncertainty estimation procedure introduces a large overhead on the hardware performance of BayesNNs, which hinders their deployment in commercial applications [4]. The uncertainty is quantified

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 and in part by the funds from Corerain, Maxeler, Intel and Xilinx. Martin Ferianc was sponsored through a scholarship from the Institute of Communications and Connected Systems at UCL.

H. Fan, Z. Que and W. Luk are with the Department of Computing, Imperial College London, London, SW7 2AZ, UK.
M. Ferianc and M. Rodrigues are with the Department of Electronic and Electrical Engineering, University College London, London, WC1E 6BT, UK.
X. Niu is with Corerain Technologies Ltd., Shenzhen, China.
* *Corresponding author: Hongxiang Fan (h.fan17@imperial.ac.uk).*

by performing N_s Monte Carlo (MC) samples that are obtained by repeatedly running the same input through the whole network, introducing a large computational overhead. In addition to increased computation, memory consumption and memory accesses also rapidly increase as BayesNN can require N_s sets of model parameters to perform the prediction as well as uncertainty quantification [8], [9]. Therefore, the computational and memory-intensive properties of BayesNNs significantly degrade their hardware performance.

In this paper, we observe that an extensive amount of structured sparsity exists in BayesNNs which can be exploited to improve the hardware performance. Different from unstructured sparsity which involves irregular zeros generated, for example, through ReLU activations [12] or pruning techniques [13], structured sparsity does not require complex hardware implementations and can be carefully exploited by proper algorithmic and hardware optimizations. We summarize the difference between structured sparsity in BayesNNs and other types of sparsity observed in standard NNs in Section 6.1. According to their characteristics, we first categorize the structured sparsity in BayesNNs into three classes: channel sparsity, layer sparsity and sample sparsity. These three categories of sparsity are controlled by three algorithmic parameters of BayesNNs respectively, i.e., the dropout rate, the number of Bayesian layers and the number of samples. Higher sparsity can reduce the amount of computation, but it may also affect various properties of BayesNNs such as their accuracy and their quality of uncertainty estimation. Therefore, we propose an automatic framework to explore the structured sparsity of BayesNNs without sacrificing their algorithmic performance. To fully exploit these three types of structured sparsity at the hardware level, a novel hardware architecture is proposed to accelerate BayesNNs, which achieves a high hardware performance using the runtime adaptable hardware engines and the intelligent skipping support. The runtime adaptability is supported by dedicated control and multiplexers, which are different from the reconfigurability provided by FPGAs. Note that the automatic framework is not limited to our proposed hardware design, but it is sufficiently general to be applicable to other hardware platforms, such as CPUs and GPUs.

A summary of our contributions is as follows.

- Exploiting structured sparsity in BayesNNs to achieve high performance. We categorize it into three classes, i.e., channel, layer and sample sparsity (Section 2.3).
- A framework that automatically explores channel, layer and sample sparsity in BayesNNs, while maintaining algorithmic performance, which improves the performance of BayesNNs on different hardware platforms (Section 3).
- A novel hardware architecture for BayesNNs with runtime adaptability and intelligent skipping optimization to achieve high performance (Section 4).
- Extensive experiments evaluating four distinct BayesNNs on four different datasets, which demonstrate the effectiveness of our hardware architecture, algorithm and optimizations (Section 5).

2 Background and Motivation

2.1 Bayesian Neural Networks

BayesNNs are making significant progress in many research areas where decision-making needs to be accompanied by uncertainty estimation [4]. By augmenting NNs with the capability of Bayesian inference, they become more robust against overfitting, even when dealing with datasets with fewer samples [10]. The principle of Bayesian inference is in learning a distribution over the weights of the NN, instead of pointwise constant estimates. The learning is performed by employing the Bayes rule and setting a prior distribution over the model class and a corresponding likelihood. Given the high-dimensionality of modern NNs [14], employing the Bayes rule to obtain the true posterior distribution over the weights or models of the NN is analytically intractable.

To address this challenge, Gal and Ghahramani [4] proposed a variational approximation, called Monte Carlo Dropout (MCD), to the true posterior which enables the use of the Bayes rule in practice. The method is built on enabling dropout [15] during evaluation as well as training with L2 regularisation resulting in Bayesian inference. Dropout randomly disconnects nodes [4] or channels [16] in an NN through a random channel-wise mask $\boldsymbol{M}_i \in \mathbb{R}^{C_i}$ to the output feature maps $\boldsymbol{Y}_i$ of layer i^{th} with C_i channels. The mask $\boldsymbol{M}_i$ follows a Bernoulli distribution $p(\boldsymbol{M}_i)$ which generates binary random variables (0 or 1) with the probability given by the dropout rate p_i. The dropout rate can be different for different layers in the Bayesian NN. After dropout removes the output feature maps with zeros. The computation for output $\boldsymbol{O}_i$ for the i^{th} layer can be formulated as $\boldsymbol{O}_i = \boldsymbol{Y}_i \boldsymbol{M}_i$. Kendall *et al.* [17] demonstrated that the Bayesian NN does not need to be Bayesian in every layer to obtain good algorithmic performance.

The uncertainty estimation and prediction are obtained by running the same input through a BayesNN N_s times, each time with a different set of sampled masks $\boldsymbol{M}$ which translate into sampling the weights from the learnt variational distribution for each layer i^{th} where dropout is applied, and averaging the outputs with respect to N_s. Hence the overall compute scales by $\mathcal{O}(N_s)$.

2.2 Structured Sparsity

Extensive research interests have been expressed in exploiting sparsity in standard NNs to improve their hardware performance. Nevertheless, most research efforts have focused on exploiting the irregular activation sparsity generated by ReLU (Rectified Linear Unit) [18] activation or weight sparsity introduced by pruning [13], [19]. An example of activation and weight sparsity is presented in Figure 2, where a large number of zeros exists in input feature maps, weights and output feature maps. To skip the redundant computation and data transfer of zeros, various hardware architectures have been proposed to support sparse convolution (SpCONV) [20], [21], [22] and sparse general matrix-matrix multiplication (SpGEMM) [23], [24], [25], [26]. However, most of them require complicated control and encoding-decoding hardware modules to manipulate and transform the compressed sparse matrix. Also, these accelerators are only effective for NNs constructed using ReLU activations, which limits their deployment in real-world

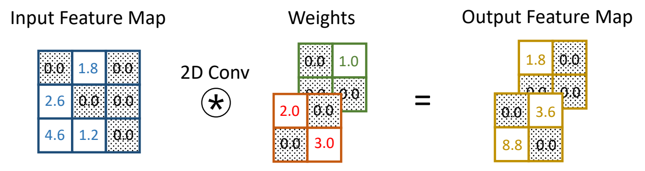


Fig. 2. Irregular activation and weights sparsity existing in 2D convolution, channel dimension is ignored for simplicity.

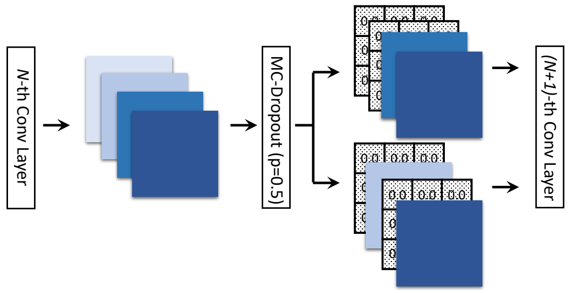


Fig. 3. Channel sparsity introduced by MCD in BayesNNs.

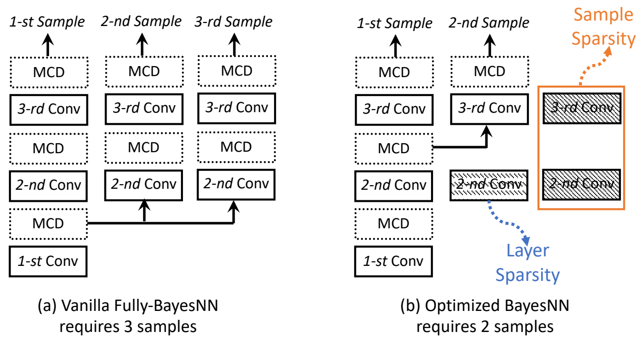


Fig. 4. Layer and sample sparsity in BayesNNs.

applications. Given that other activation functions such as leaky ReLU [27], [28] have been widely adopted in various NNs due to the continuous development of deep learning, there is a need for other techniques to improve the hardware performance of general NNs.

Different from the standard sparse NNs that only contain the irregular activation and weight sparsity, we observe that extensive structured sparsity exists in BayesNNs. This paper categorizes the structured sparsity of BayesNN into three classes: channel sparsity, layer sparsity and sample sparsity.

2.2.1 *Channel Sparsity*

The channel sparsity in this paper refers to the channels dropped out by the MCD in both input and output feature maps [16]. Figure 3 presents an example of the channel sparsity in BayesNNs. Receiving the output feature maps from the previous convolutional (CONV) layer, the MCD randomly drops out half of channels with a dropout rate $p = 0.5$. As a result, there is 50% channel sparsity existing in the input feature maps for the next CONV layer. The dropout rate decides the channel sparsity, which further affects both algorithmic and hardware performance. Although a higher dropout rate exhibits the higher channel sparsity that can be exploited during hardware acceleration for better hardware performance, it may also degrade the algorithmic performance. In this paper, we systematically explore this algorithmic and hardware performance trade-off by optimizing the dropout rate for each layer by a thorough design space exploration, introduced in Section 3.

2.2.2 *Layer Sparsity*

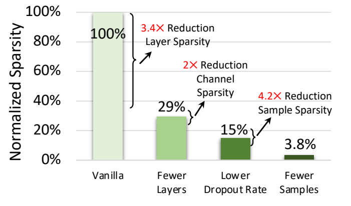
In BayesNN, it requires running the Bayesian layers, i.e., the layers followed by MCD, multiple times to make the uncertainty estimation. One question raised during this process is how many Bayesian layers are required to obtain the uncertainty estimation. Previous work has partially addressed this question [17], [29], indicating that making different parts of the network Bayesian can improve uncertainty estimation and prediction accuracy. In this paper, we refer to the non-Bayesian layers as the layer sparsity in BayesNNs, which is illustrated in Figure 4. In contrast to the regular fully-BayesNNs, the optimized BayesNN only has the last CONV layer as Bayesian, which eliminates the requirement of running the first and second CONV layers with other samples. The redundant computation of the first and second CONV layers is the layer sparsity that we are going to exploit in this paper.

2.2.3 *Sample Sparsity*

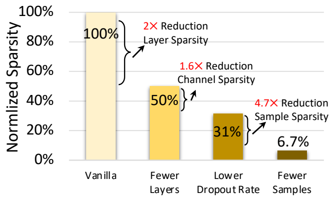
Another question raised is: how many MC samples N_s are required to achieve satisfactory algorithmic performance? In our experiments, we observe diminishing returns in terms of the observed algorithmic performance while increasing N_s. Limiting the N_s to the lowest number is denoted as sample sparsity in our paper. Figure 4 illustrates the sample sparsity in BayesNNs. In the optimized three-layer BayesNN, it only requires two samples to achieve the same algorithmic performance as the fully-BayesNN with three samples. The redundant third sample in the regular fully-BayesNN is referred to as the sample sparsity.

2.3 Motivation

To quantitatively analyze the effect of the channel, layer and sample sparsity, a sparsity breakdown of *Bayes-AlexNet* and *Bayes-VGG11* is presented in Figure 5. We used *MNIST* [30] and *SVHN* [31] datasets for *Bayes-AlexNet* and *Bayes-VGG11* respectively. For each model, we evaluated both vanilla and sparse versions for comparison. Their algorithmic settings and performance are summarized in Table 1. We refer to the vanilla version to be the fully-Bayesian NN with MCD applied after every CONV layer, which reflects the original settings [4]. For the sparse version, we randomly chose the one with similar algorithmic performance while using optimized dropout rates, fewer Bayesian layers and MC samples for demonstration. Apart from the classification accuracy (Acc), we also measured the predictive uncertainty and confidence. We measured the uncertainty expressiveness of the Bayesian architectures through observing the average predictive entropy (aPE) over a dataset of size E as: aPE $= \frac{1}{E}\sum_{e=1}^{E} - \sum_{k=1}^{K} p(y_e^k|\boldsymbol{x}_e)\log p(y_e^k|\boldsymbol{x}_e)$. The K is the number of output classes, whereas $\boldsymbol{x}$ and $\boldsymbol{y}$ represent the input output pairs. We measured aPE with respect to random Gaussian noise with mean and standard deviation of the training data that should rightfully



(a) Sparsity break down of Bayes-AlexNet



(b) Sparsity break down of Bayes-VGG11

Fig. 5. Breakdown of three classes of structured sparsity in BayesNNs.

TABLE 1
Breakdown of three classes of structured sparsity in BayesNNs.

Model	# of Bayes Layers	# of Samples	Dropout rate	Error (%)	aPE (nats)	ECE (%)
Baseline *Bayes-AlexNet*	7	100	0.125	0.88	1.374	0.149
Sparse *Bayes-AlexNet*	4	20	0.5	0.78	1.592	0.127
Baseline *Bayes-VGG11*	10	100	0.125	3.51	2.001	0.410
Sparse *Bayes-VGG11*	7	20	0.375	3.81	2.100	0.348

confuse the net and result in high value of entropy. Additionally, we measured the confidence of the Bayesian architectures on the test data using the expected calibration error (ECE) [32]. ECE computes a weighted average of a mismatch between confidence and error rate across bins as: ECE $= \sum_{b=1}^{B} \frac{n_B}{E} |\text{accuracy}(b) - \text{confidence}(b)|$, where n_b is the number of predictions in bin b and accuracy(b) and confidence(b) are the accuracy and confidence of bin b, respectively. We set $B = 10$.

As we can see from Figure 5, both *Bayes-AlexNet* and *Bayes-VGG11* encompass over 93% of sparsity compared to their vanilla counterparts with similar or better algorithmic performance. Specifically, channel, layer and sample sparsity provides $2\times$, $3.4\times$ and $4.2\times$ reduction in the total amount of calculation for *Bayes-AlexNet*, and $1.6\times$, $2\times$ and $4.7\times$ less computation for *Bayes-VGG11*. As higher sparsity leads to better performance for our design, an automatic framework (Section 3) is proposed to increase the structured sparsity of BayesNNs while maintaining the algorithmic performance. Our framework can be also applied on other hardware platforms such as CPU and GPU to improve their hardware performance. To exploit the extensive sparsity in BayesNNs, this paper proposes a novel hardware architecture to effectively skip the structured zeros caused by channel sparsity, layer sparsity and sample sparsity (Section 4).

3 Optimization Framework

3.1 Framework Overview

An overview of our proposed framework is presented in Figure 6. The input search space, bounded by the user, is defined by the algorithmic network design space. Given an user-supplied network structure, the network design space contains all potential BayesNN architectures with different dropout rates and number of Bayesian layers that can be obtained by modifying the user-supplied architecture.

The inputs also contain the input dataset and the description of the target application and the device. Given the specification of the input constraints, the framework aims at exploring the structured sparsity of BayesNNs to improve both algorithmic and hardware performance. Note that, the optimized BayesNN will also be trained during this process. The outputs of the framework are the optimized algorithmic parameters, which includes the number of Bayesian layers, dropout rate of each Bayesian layer and the number of MC samples N_s. The final optimized BayesNN can then be deployed for the target applications such as medical imaging or self-driving.

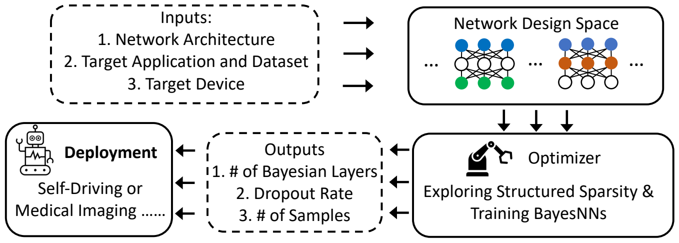


Fig. 6. Overview of the optimization framework.

3.2 Algorithm for Exploring Structured Sparsity

Due to the large design space and expensive training cost, we present an algorithm for exploring structured sparsity. The proposed algorithm contains four phases: obtaining the performance baseline, exploring layer sparsity, increasing channel sparsity and exploiting sample sparsity. Algorithm 1 presents the pseudocode.

In the first phase, we train the vanilla BayesNNs with MCD applied after every convolutional layer with an uniform dropout rate, which adheres to the same practice as in [4]. We iterate through different dropout rates $\{0.125, 0.25, 0.375, 0.5\}$ and choose the one with the best algorithmic performance, getting $Acc^{'}$, $ECE^{'}$ and $aPE^{'}$ (Section 2.3) as the algorithmic performance baselines. The goal of the next three phases is to explore the structured sparsity for a higher hardware performance while achieving a similar algorithmic performance as the baseline.

In the second phase, we explore the layer sparsity by optimizing the number of Bayesian layers. We train the BayesNNs with different number of Bayesian layers and then choose the one with the best algorithmic performance. After getting the optimized number of Bayesian layers n^{opt}, we then explore the channel sparsity in the third phase.

As higher dropout rates have a higher channel sparsity, the third phase attempts to increase the dropout rate while not sacrificing the algorithmic performance. The hill-climbing algorithm is adopted, which increases the dropout rate by 0.125 in each step until the performance converges. An optimized dropout list dr_list^{opt} is obtained after the third phase.

The last phase exploits the sample sparsity by decreasing the number of samples N_s. This phase evaluates the BayesNN using n^{opt}, dr_list^{opt} with different N_s. The measured performance is compared against the baseline performance $Acc^{'}$, $ECE^{'}$ and $aPE^{'}$. We set the accuracy threshold δ as 0.3% to allow negligible accuracy loss.

Algorithm 1 Algorithm for Exploring Structured Sparsity.

```
1: Phase 1: Getting the performance baseline
2: $N^l = \{N^l_1, N^l_2, N^l_3, N^l_4\}$    ▷ Number of Bayesian layers
3: $dropout\_rates = \{0.125, 0.25, 0.375, 0.5\}$
4: $Acc^{'} = 0.0, ECE^{'} = 0.0, aPE^{'} = 0.0$    ▷ Algorithmic metrics
5: For $dr$ in $dropout\_rates$:
6:     $Acc, ECE, aPE$ = Train_Full_Bayes($dr$)
7:     If ($Acc > Acc^{'}$ and $ECE > ECE^{'}$ and $aPE > aPE^{'}$):
8:         $Acc^{'} = Acc, ECE^{'} = ECE, aPE^{'} = aPE$
9: Phase 2: Exploring layer sparsity
10: $Acc^{opt} = 0.0, ECE^{opt} = 0.0, aPE^{opt} = 0.0$
11: $n^{opt} = N^l_1, dr^{opt} = 0.125$
12: For $dr$ in $dropout\_rates$:
13:     For $n$ in $N^l$:
14:         $Acc, ECE, aPE$ = Train_Uniform_Bayes($dr, n$)
15:         If ($Acc > Acc^{opt}$ and $ECE > ECE^{opt}$ and $aPE > aPE^{opt}$):
16:             $n^{opt} = n, dr^{opt} = dr$
17:             $Acc^{opt} = Acc, ECE^{opt} = ECE, aPE^{opt} = aPE$
18: Phase 3: Increasing channel sparsity
19: $dr\_list[n^{opt}] = [dr^{opt}] \times n^{opt}$    ▷ Dropout rate of each layer
20: While(True):    ▷ Hill climbing optimization
21:     $j = 0$
22:     For $i$ in $(1, n^{opt})$:
23:         $dr\_list[i] += 0.125$
24:         $Acc, ECE, aPE$ = Train_Bayes($dr\_list, n^{opt}$)
25:         If ($Acc > Acc^{opt}$ and $ECE > ECE^{opt}$ and $aPE > aPE^{opt}$):
26:             $j = i$
27:             $Acc^{opt} = Acc, ECE^{opt} = ECE, aPE^{opt} = aPE$
28:         $dr\_list[i] -= 0.125$
29:     if ($j == 0$): break
30: Phase 4: Exploiting sample sparsity
31: $S = \{s_1, s_2, s_3, s_4\}, s^{opt} = \infty$
32: For $s$ in $S$:
33:     $Acc, ECE, aPE$ = Eval_Bayes($dr\_list, s$)
34:     If ($Acc > (Acc^{'} + \delta)$ and $ECE > ECE^{'}$ and $aPE > aPE^{'}$):
35:         If ($s^{opt} > s$): $s^{opt} = s$
```

4 Hardware Accelerator

4.1 Hardware Architecture

4.1.1 Architecture Overview

A design overview of our FPGA-based hardware accelerator is presented in Figure 7(a). The core computational module is a stack of processing engines (PEs) in the bottom right corner, shown in gray. To feed the inputs and weights into PEs and control the overall dataflow, the design uses different managers. The input and weight Bernoulli samplers are designated to implement MCD. To avoid a large on-chip memory consumption, all the intermediate results between layers are transferred back to the off-chip memory through DMA in parallel. Only the inputs and weights of the current processing layer are cached in the on-chip memory to improve the data locality and ease the bandwidth requirement. The computation of the whole BayesNNs is performed layer-by-layer using the same PEs. In this paper, we design both PEs and Bernoulli samplers with runtime adaptability to achieve a higher hardware performance.

4.1.2 PE with Adaptable Connectivity

The computation of convolution is essentially carried out by six nested loops in H (height of input feature maps), W (width of input feature maps), F (number of output feature maps), C (number of input feature maps), I (kernel height) and J (kernel width) dimensions. Therefore, loop unrolling can be applied in six dimensions, which leads to six unrolling factors: $\langle T_w, T_h, T_f, T_c, T_i, T_j\rangle$. Different unrolling and parallelism strategies have been proposed in previous work, such as synapse parallelism with $\langle T_i, T_j\rangle$, neuron parallelism with $\langle T_h, T_w\rangle$ and feature map parallelism with $\langle T_f, T_c\rangle$ [12]. In this paper, we propose a hybrid parallelism strategy to leverage the different parallelism combinations with an adaptable PE design.

To eliminate the memory consumption of caching the intermediate outputs of the current processing layer, we perform the nested loops of convolution in a sequence $\langle F \rightarrow H \rightarrow W \rightarrow I \rightarrow J \rightarrow C\rangle$ such that the intermediate results after the accumulation can be transferred back to the off-chip memory directly without caching on-chip. As most convolutional layers exhibit a higher concurrency in F and C dimensions, we exploit the parallelism in these two dimensions with $\langle T_f, T_c\rangle$. Therefore, we deploy N_{pe} PEs to process multiple filters in parallel, i.e., making $T_f = N_{pe}$. Within each PE, there are N_{mult} multipliers followed by a $log_2 N_{mult}$-level adder tree, which are used to compute multiple channels in parallel, i.e., making $T_c = N_{mult}$.

However, we observe that many channels are dropped out by MCD in BayesNNs, which leads to insufficient channel concurrency in some layers. For instance, the second convolutional layer in *ResNet50* has only 32 valid channels after the MCD is applied using a $p = 0.5$ dropout rate. Therefore, the computational resources allocated for channel parallelism may not be fully utilized in these cases. To address this issue, we propose a hybrid parallelism strategy for BayesNNs.

When the channel concurrency is insufficient, we reuse the computational resources to exploit parallelism in W dimension, which leads to $\langle T_w, T_f, T_c\rangle$ parallelism. To support that, we design the MAC unit in our PE with a runtime adaptation using multiplexers and demultiplexers as shown in Figure 7(b). While processing some layers with small number of valid channels, the MAC unit is grouped as T_w sub-trees to process multiple data points along the W dimension, which adopts a new parallelism strategy $\langle T_w, T^{'}_f, T_c\rangle$ with $T^{'}_f = \frac{T_f}{T_w}$. With the hybrid parallelism strategy, our design is able to achieve a better resource efficiency while processing different BayesNNs with different algorithmic features, mainly the dropout rates on a per-layer granularity.

4.1.3 Adaptable Bernoulli Sampler

There are two Bernoulli samplers in our design, which are used to perform MCD on inputs and outputs respectively. As mentioned in Section 2.1, instead of using a uniform dropout rate across the entire Bayesian NN, this paper optimizes the dropout rate for each layer, i.e., the layer-wise dropout rate. Therefore, the target design is required to perform Bernoulli sampling with arbitrary dropout rates. As the common dropout rates for MCD are 0.125, 0.25, 0.375 and 0.5 [4], [15], [17], we propose an adaptable Bernoulli sampler supporting four different probabilities using one unified hardware design.

Figure 7(c) presents the hardware design of the adaptable Bernoulli sampler. A 4-tap linear feedback shift register (LFSR) module is used to generate a single bit pseudo random variable. Three independent LFSR modules are concatenated together to generate a 3-bit random variable v, which follows a discrete uniform distribution $P(v = i) = \frac{1}{2^3}$ for $v \in \{000, 001, \ldots, 111\}$. Then, a comparator is placed

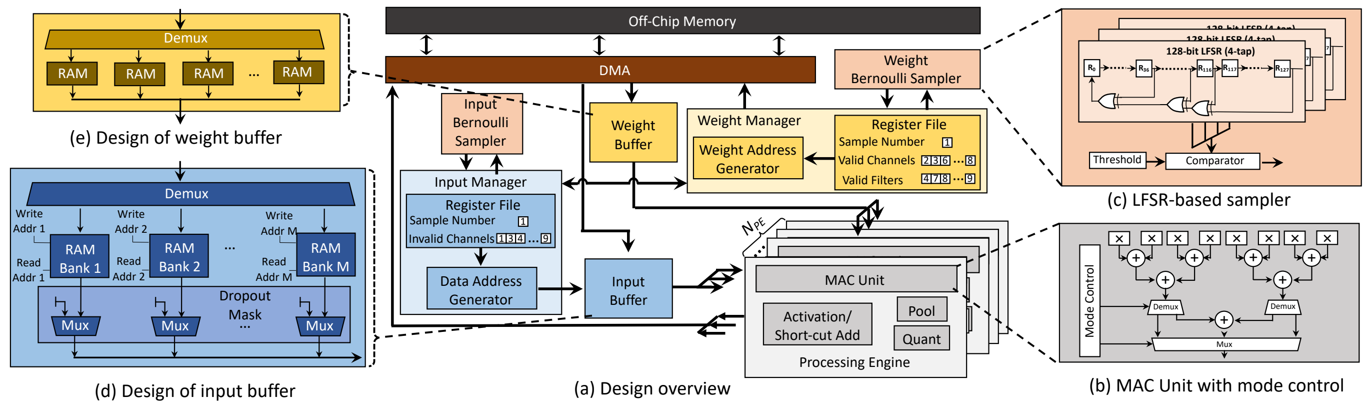


Fig. 7. Overview and the individual modules of the hardware design.

after LFSRs to produce Bernoulli variables by comparing the generated 3-bit random variable with a threshold. Note that, the threshold is held in a register and can be configured according to the required probability, which makes our Bernoulli sampler adaptable for different dropout rates. For instance, while performing the Bernoulli sampling with probability 0.375, the threshold is set as 010 to compare with the generated 3-bit random variables.

4.1.4 *Inputs Control*

The input dataflow is controlled by an input manager and an input buffer. The input manager is composed of an address generator and a register file that receives Bernoulli variables from the Bernoulli sampler indicating the dropped out channels. The input buffer mainly consists of a dropout mask and T_c RAM banks. The dropout mask includes T_c multiplexers, which sets the corresponding channels to zeros when the MCD is enabled. Each RAM uses separate read and write addresses generated from the input address generator. The T_c data points generated from the RAM banks are duplicated N_{pe} copies which then flow into PEs.

The input data in the RAM banks are stored in a $\langle W$, H, $C\rangle$ sequence, which is illustrated in Figure 8. Different channels that belong to the same spatial position, i.e., the same height and weight, are first stored evenly among T_c RAM banks. Since T_c is less than C, it takes C/T_c columns to store all different channels. Then, the data are stored along the width dimension from H_1W_1 to H_1W_n. The inputs along the height dimension are stored in last. When T_c is less than the number of channels, the data from different width will be stored first, which facilitates the data access of our hybrid parallelism strategy $\langle T_w/T_f$, $T_c\rangle$. Since each RAM bank has separate read and write addresses, the data address generator is designed to support sequential and line-buffer modes. The sequential mode reads and writes sequentially from and into RAM banks, which is required by some layers such as the fully connected (FC) layer. The line-buffer mode allows the reading and writing to be performed in a sliding window manner, which captures the data access pattern of 2D convolutions.

4.1.5 *Weight Control*

The weights are controlled by a weight manager and a weight buffer. The weight buffer contains T_f RAM banks to cache weights from different filters. To reduce the on-chip memory consumption, we only cache T_f filters of weights on-chip for the current processing layer. The double buffer technique is adopted to overlap the weights transfer with the computation [33]. The bit width of each RAM bank in weight buffer is $T_f \times DW$ with DW denoted as data width, which is used to store weights from different channels in the same filter. The weights are stored in the RAM banks in a $\langle F, I, J, C\rangle$ sequence as shown in Figure 9, which facilitates the weight access for the $\langle F \rightarrow H \rightarrow W \rightarrow I \rightarrow J \rightarrow C\rangle$ computational order.

The weight manager contains a register file and a weight address generator. While processing the non-Bayesian layers, the weight address generator produces the read addresses sequentially. When the design is processing the Bayesian layers, the read address is generated according to the valid channel and filters in the register file. The generated addresses are then fed into the DMA controller to transfer the required weights from off-chip memory to the on-chip weight buffer. In off-chip memory, the weights are also stored in a $\langle F, I, J, C\rangle$ sequence layer-by-layer.

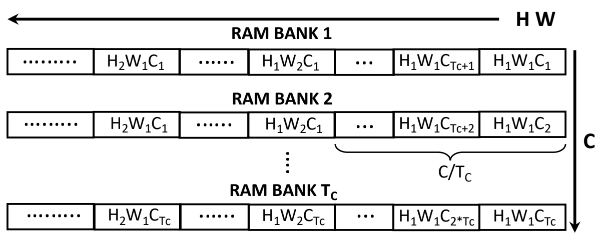


Fig. 8. The storage of input data in RAM banks.

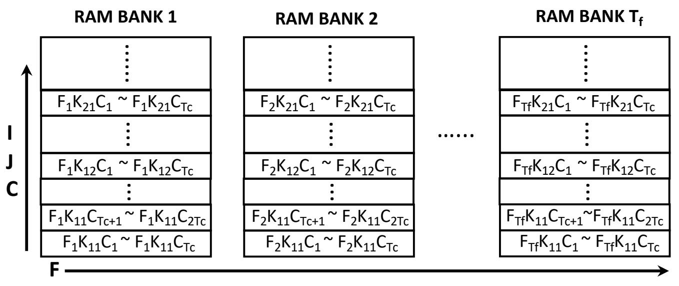

RAM BANK 1
RAM BANK 2
RAM BANK T_f
$F_1K_{21}C_1$ ~ $F_1K_{21}C_{Tc}$
$F_1K_{12}C_1$ ~ $F_1K_{12}C_{Tc}$
$F_1K_{11}C_{Tc+1}$ ~ $F_1K_{11}C_{2Tc}$
$F_1K_{11}C_1$ ~ $F_1K_{11}C_{Tc}$
$F_2K_{21}C_1$ ~ $F_2K_{21}C_{Tc}$
$F_2K_{12}C_1$ ~ $F_2K_{12}C_{Tc}$
$F_2K_{11}C_{Tc+1}$ ~ $F_2K_{11}C_{2Tc}$
$F_2K_{11}C_1$ ~ $F_2K_{11}C_{Tc}$
$F_{Tf}K_{21}C_1$ ~ $F_{Tf}K_{21}C_{Tc}$
$F_{Tf}K_{12}C_1$ ~ $F_{Tf}K_{12}C_{Tc}$
$F_{Tf}K_{11}C_{Tc+1}$ ~ $F_{Tf}K_{11}C_{2Tc}$
$F_{Tf}K_{11}C_1$ ~ $F_{Tf}K_{11}C_{Tc}$
I J C
F

Fig. 9. The storage of weights in RAM banks.

4.2 Intelligent Skipping Optimization

To exploit the extensive structured sparsity in BayesNNs, we design our hardware to skip unnecessary samples, layers and channels.

4.2.1 Skip Samples

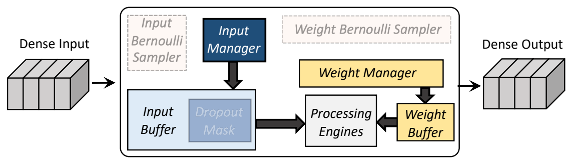
Skipping MC samples is straightforward to implement in our hardware. In both data and weight register files, we design a register storing the number of MC samples N_s for the current processing layer. Together with the optimization framework, introduced in Section 3, our design can efficiently reduce the overall latency of BayesNNs. For instance, after the framework decreases the number of samples from 100 to 5, we set the number of samples in both data and weight register files as 5. Then the design only needs to run each layer for 5 times to produce the uncertainty estimate.

4.2.2 Skip Layers

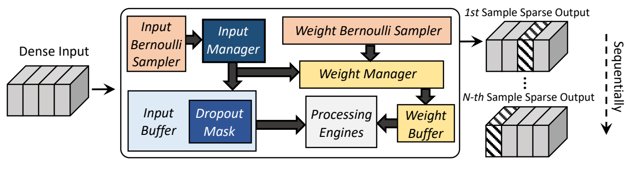
To estimate uncertainty and obtain the prediction, it is required to run the whole BayesNN multiple times. However, as the intermediate results of the non-Bayesian layer in different samples are the same, the redundant computation related to these non-Bayesian layers can be skipped [1], [34]. To achieve this, our proposed design supports three modes, i.e., non-dropout, dual-dropout and output-dropout modes, to run different layers in BayesNNs.

While running the non-Bayesian layers, the design adopts the non-dropout mode, which disables the dropout mask and both input and weight Bernoulli samplers as illustrated in Figure 10(a). To further avoid redundant computation, we also apply non-dropout mode on the first Bayesian layer, which generates the dense results so that the output can be reused in the second Bayesian layer. Note that, the MCD of the first Bayesian layer will be performed together with the second Bayesian layer using dual-dropout mode. All the computation under the non-dropout mode will be only performed once, which significantly reduces the overall latency.

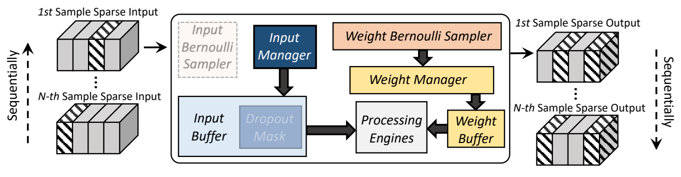
The design keeps the non-dropout mode to process the network layer-by-layer until the second Bayesian layer, and then turns into dual-dropout mode by enabling the dropout mask, input and weight Bernoulli samplers as shown in Figure 10(b). In the dual-dropout mode, the inputs will be cached in the input buffer and reused to perform the computation required by different samples. The sample results are produced sequentially and transferred back to the off-chip memory. The dual-dropout mode also applies the MCD in both inputs and outputs as the MCD of the previous, i.e., the first, Bayesian layer is not performed yet. The MCD of the inputs is implemented by a dropout mask in the input buffer, while the MCD on the outputs is performed using the weight Bernoulli sampler and the weight manager. For the rest of Bayesian layers, the design is configured as an output-dropout mode with only the weight Bernoulli sampler enabled, which is illustrated in Figure 10(c). The output-dropout mode will only apply MCD on the outputs by using the weight Bernoulli sampler and weight manager. By executing different modes for different layers, our design can effectively skip the redundant computation of non-Bayesian layers, while keeping the ability to produce the uncertainty estimate.



(a) Non-dropout mode for Non-Bayesian layers and the first Bayesian layers

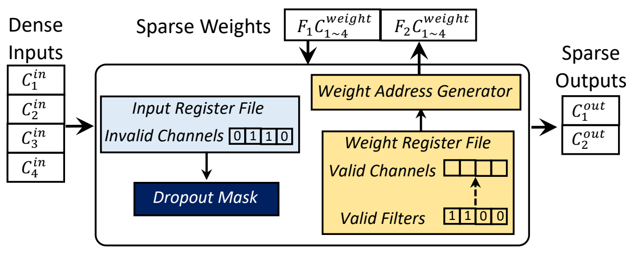


(b) Dual-dropout mode for the second Bayesian layer

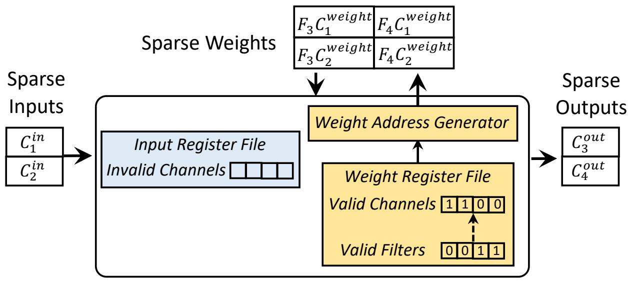


(c) output-dropout mode for the rest of Bayesian layers

Fig. 10. Hardware execution under different modes.



(a) Skip Chanels in dual-dropout mode



(b) Skip Chanels in output-dropout mode

Fig. 11. Register status while skipping channels.

4.2.3 Skip Channels

We design our hardware to support channel skipping in both dual-dropout and output-dropout modes by dedicated controls and a set of registers in the register file. Figure 11 illustrates an example of running a Bayesian layer with 4 channels and 0.5 dropout rate on our design. In dual-dropout mode, the design receives a dense input with four channels ($C^{in}_{1\sim4}$). We then apply the MCD on inputs using the dropout mask, which sets some channels as zeros according to the invalid channel indexes in the input register file. The channel skipping is then implemented by the weight manager and the weight register file. Given a register with a 4-bit value $\{1,\ 1,\ 0,\ 0\}$ indicating the valid filters, the weight address generator produces the read address to transfer only the first and second filters of weights from off-chip memory into on-chip weight buffer. Therefore, the computation for the third and fourth output filters can be avoided. Note that, as the inputs are dense, the addressing of weights from the weight address generator is only based on the valid filters. In output-dropout mode as shown in Figure 11, the weight addressing is based on both valid channels and filters. For instance, given the registers of valid channels and filters to be $\{1,\ 1,\ 0,\ 0\}$ and $\{0,\ 0,\ 1,\ 1\}$, only the first and second channels in the third and fourth filters ($F_3C_{1\sim 2}^{weight}$ and $F_4C_{1\sim 2}^{weight}$) will be loaded into on-chip memory instead of all the channels, which further eases the bandwidth requirement.

5 Evaluation

5.1 Experimental Setup

We implemented our hardware design using Verilog on an Intel Arria 10 SX660 platform with a 1GB DDR4 SDRAM installed as an off-chip memory. Quartus 17 Prime Pro was used for synthesis and implementation and the final design was clocked at 222 MHz. The 8-bit linear quantization [35], [36] was adopted in our design to improve the hardware performance. We used one DSP with some extra logic resources to implement two multipliers to save DSP resources. We optimized T_f, T_w, T_c and the PE mode introduced in Section 4 according to the total amount of available resources in the underlying hardware platform and the available concurrency exhibited in running BayesNNs. The final optimized FPGA design consumed $1,492$ DSPs, $2,432$ M20Ks, $303,913$ ALMs and $889,869$ registers. To demonstrate the effectiveness and generality of our hardware design and automatic framework for accelerating BayesNNs, we evaluated four BayesNNs, including *Bayes-VGG11*, *Bayes-AlexNet*, *Bayes-ResNet18* and *Bayes-ResNet50*, on four different datasets for image classification, i.e., SVHN [31], MNIST [30], CIFAR-10 and CIFAR-100 [37]. The dropout rate was selected from $\{0.125, 0.25, 0.375, 0.5\}$. We chose the number of Bayesian layers for *Bayes-VGG11* from $\{10, 7, 5, 3, 1\}$, *Bayes-AlexNet* from $\{7, 5, 3, 2, 1\}$, *Bayes-ResNet18* from $\{21,\ 16,\ 11,\ 6,\ 1\}$, *Bayes-ResNet50* from $\{54,\ 44,\ 25,\ 12,\ 1\}$. The number of samples N_s was selected from $\{5,\ 10,\ 20,\ 50,\ 100\}$. The hardware performance was evaluated in terms of latency, energy consumption and energy efficiency. We measured the algorithmic performance using classification error, ECE and aPE as detailed in Section 2.3.

5.2 Framework Effectiveness and Exploiting Sparsity

Following the detailed procedures for exploring the structured sparsity in Section 3.2, we evaluated the improvement brought by each type of sparsity in our FPGA design. While optimizing a single BayesNN, the training settings stay the same during the whole process as different variants of the same BayesNN have similar convergence time. The whole optimization process took 30∼150 GPU hours to complete depending on the neural architectures and datasets.

5.2.1 Exploiting Layer and Channel Sparsity

We obtained the baseline performance for all BayesNNs by following the first phase of Algorithm 1. The baseline performance was derived using the maximum $N_s = 100$ MC samples to get the best approximation to the achievable algorithmic performance. Table 2 presents the baseline performance of *Bayes-VGG11*, *Bayes-AlexNet*, *Bayes-ResNet18* and *Bayes-ResNet50* with their corresponding configurations. As it can be seen, all the BayesNNs adopted 0.125 as the uniform dropout rate to achieve higher algorithmic performance.

TABLE 2
The resultant configurations of BNNs.

Model	Version	# of Bayes Layers	Dropout Rate	Error (%)	aPE (nats)	ECE (%)
Bayes-VGG11	Baseline	10	$\mathbf{0.125}_{1\sim 10}$	3.511	2.004	0.410
	Optimized	7	$\mathbf{0.375}_{4\sim 7,9\sim 10}$ $\mathbf{0.5}_{8}$	3.776	2.217	0.297
Bayes-AlexNet	Baseline	7	$\mathbf{0.125}_{1\sim 7}$	0.880	1.374	0.149
	Optimized	5	$\mathbf{0.5}_{3\sim 7}$	0.889	1.837	0.138
Bayes-ResNet18	Baseline	21	$\mathbf{0.125}_{1\sim 21}$	6.63	1.093	3.058
	Optimized	16	$\mathbf{0.125}_{7\sim 9,11\sim 16,18\sim 21}$ $\mathbf{0.25}_{6,10,17}$	6.580	1.580	1.202
Bayes-ResNet50	Baseline	54	$\mathbf{0.125}_{1\sim 54}$	23.700	1.405	2.109
	Optimized	44	$\mathbf{0.125}_{12\sim 31,35\sim 54}$ $\mathbf{0.25}_{32\sim 34}$	21.520	1.481	0.904

Then, to evaluate the effect of exploiting layer and channel sparsity, we applied the second and third phases of Algorithm 1 to the BayesNNs. The optimized number of layers and dropout rates of each BayesNN are presented in Table 2. The subscript of the dropout rate denotes the position of the Bayesian layers. We also evaluated the algorithmic performance of these optimized BayesNNs, which is shown again in Table 2. To eliminate the influence of sample sparsity, the performance was measured again with respect to $N_s = 100$ MC samples. As it can be observed, the optimized BayesNNs achieved better classification accuracy, ECE and aPE than the baseline versions. Even though there is nearly a 0.25% accuracy loss on *Bayes-VGG11*, aPE was improved by 0.213 and the ECE decreased by 0.113. We also present the normalized layer and channel sparsity for the four BayesNNs in a bar chart in Figure 12. As it can be seen, the BayesNNs achieved $18 \sim 49\%$ layer sparsity and $9 \sim 26\%$ channel sparsity.

To demonstrate the effectiveness of our hardware architecture in exploiting the layer and channel sparsity by supporting layer and channel skipping, we evaluated the optimized BayesNNs on our design. The speedup breakdown of the four BayesNNs is presented in Figure 12. It can be clearly seen from the bar chart on the right of each model that our design achieved 1.2∼$2\times$ speedup by exploiting the layer sparsity across different BayesNNs. Furthermore, by using the dedicated hardware design for layer skipping, another $1.9\times$ speedup can be obtained and $2\times$ speedup can be gained by exploiting the channel sparsity, specifically in *Bayes-VGG11* and *Bayes-AlexNet* respectively. Because of the limited channel sparsity in *Bayes-ResNet18* and *Bayes-ResNet50*, skipping channels can only reduce the latency by 1.15 and 1.16 times.

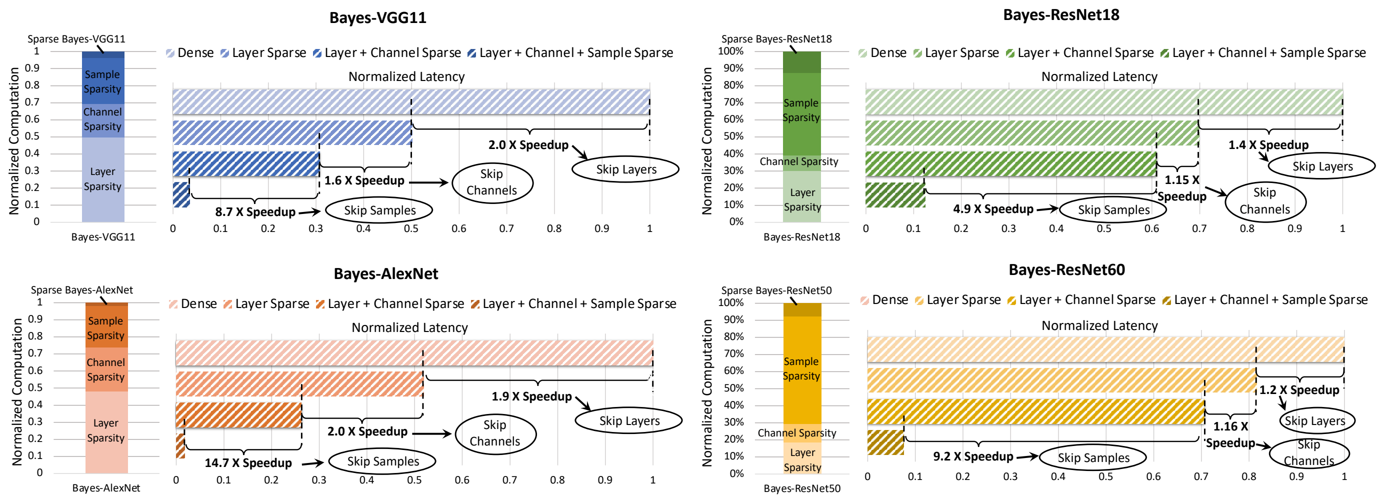


Fig. 12. Sparsity percentage and speedup breakdown on *Bayes-VGG11*, *Bayes-AlexNet*, *Bayes-ResNet18* and *Bayes-ResNet50*.

5.2.2 *Exploiting Sample Sparsity*

After determining the optimized Bayesian layers and dropout rates, we further exploited the sample sparsity using the last phase of Algorithm 1. To visualize the effect of the number of samples on the algorithmic performance, we evaluated the four different optimized BayesNNs by setting N_s to 5, 10, 20, 50 and 100 MC samples. The evaluation was repeated 5 times using different random seeds. We present the results in Figure 13 with both mean and standard deviation. As it can be seen, the aPE shows an increasing trend when the number of samples becomes larger. The classification error shows a steady decrease when the number of samples increases. By comparing the measured performance of optimized BayesNNs against the baseline performance, we choose the number of samples to be 10, 5, 20 and 10 for *Bayes-VGG11*, *Bayes-AlexNet*, *Bayes-ResNet18* and *Bayes-ResNet50* respectively. To quantitatively evaluate the effect of sample sparsity, Figure 12 presents the normalized sample sparsity and the corresponding speedup achieved on our design. By using Algorithm 1 to explore structure sparsity, it can achieve $24\%{\sim}63\%$ sample sparsity varying from different BayesNNs. Our design also reduced the latency by $4.9{\sim}14.7\times$ on different BayesNNs. Together with layer and channel sparsity, the overall structured sparsity addressed by our framework ranges from 87% to 97%. At the same time, optimized BayesNNs on our FPGA design can achieve $6.5{\sim}56\times$ speedup.

5.3 Improvement on CPU and GPU

The algorithm optimization in our framework can also be applied to other hardware platforms to improve performance. We applied our framework on an Intel Xeon E5-2680 v2 CPU and a Nvidia GeForce RTX 2080 Ti GPU. We used PyTorch [38] for both CPU and GPU implementations as it is an Nvidia-optimized deep learning framework adopted by the MLPerf benchmark [39]. To provide a fair comparison, various optimization techniques were enabled, such as cuDNN and OpenMP[1]. As PyTorch did not support skipping channels, we only enabled the skipping of samples and layers on CPU and GPU. The skipping of samples was implemented by controlling the loop variables during evaluation and the skipping of layers was supported by caching the intermediate results of non-Bayesian layers [1], [40]. The results are presented in Figure 14. As it can be observed, the optimized BayesNNs reduce the latency by $6.3{\sim}49.3$ times on the CPU implementation and $6.1{\sim}40.4$ times on the GPU implementation, which demonstrates the generality of our framework. Note that, our framework can be applied to any hardware accelerator with the support of layer, channel and sample skipping to improve the hardware performance.

5.4 Comparison of FPGA and GPU

To demonstrate the advantages of our hardware design in accelerating BayesNNs, we compared the performance of our FPGA design against the GPU implementation. The performance metrics of GPU implementations were kept the same as in the previous work [12], [40], [41]. *Bayes-VGG11* was selected for comparison as it represents the type of NN constructed by using common regular 2D convolutions with small kernel sizes. We also evaluated *Bayes-ResNet50* as it contains residual connections. The GPU implementation was the same as in Section 5.3 with sample skipping (Opt-S) and layer skipping (Opt-L) enabled. On FPGA designs, apart from using both sample and layer skipping, we also implemented with and without channel skipping (Opt-C). The results are presented in Table 3. As we can see from the table, our hardware architecture on an FPGA with all optimization applied can achieve 7.6 and 7.1 times speedup on *Bayes-VGG11* and *Bayes-ResNet50* compared with the GPU implementation. Besides, we were also able to achieve 39.3 and 37.1 times higher energy efficiency on the *Bayes-VGG11* and *Bayes-ResNet50* respectively. These gains of our design were mainly achieved through:

- A deep pipelined design with channel skipping, which decreases the memory traffic between consecutive layers.
- The algorithm optimization of the layer-wise dropout rates, which maximally increases the structured sparsity while maintaining the hardware performance.

1. https://pytorch.org/tutorials/recipes/recipes/tuning_guide.html

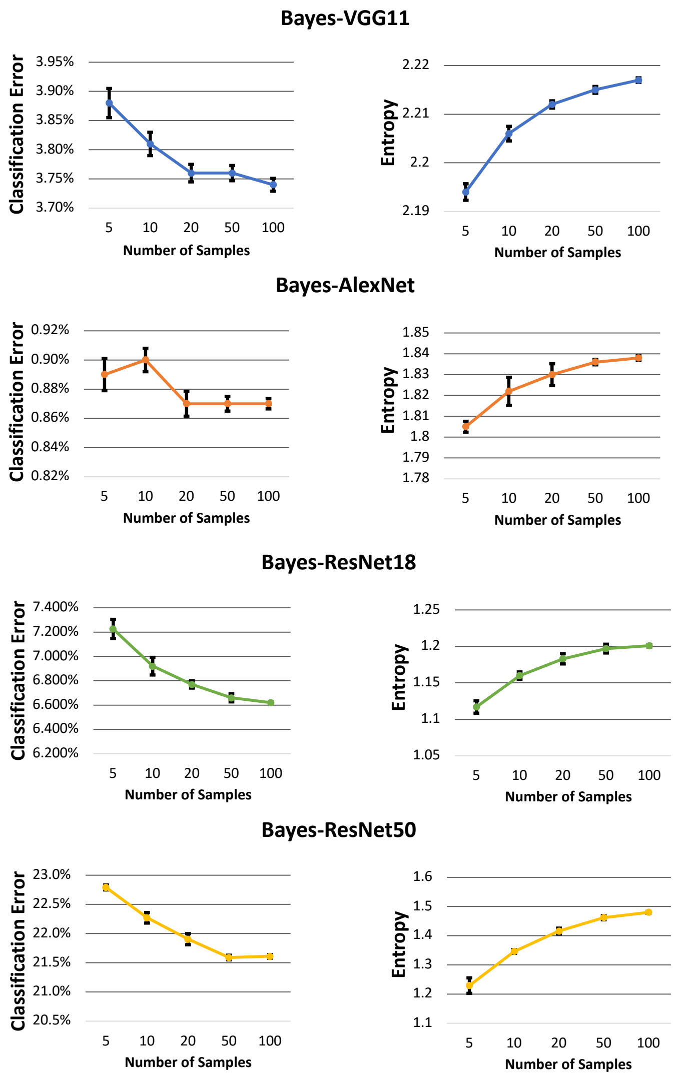


Fig. 13. Effect of the number of samples.

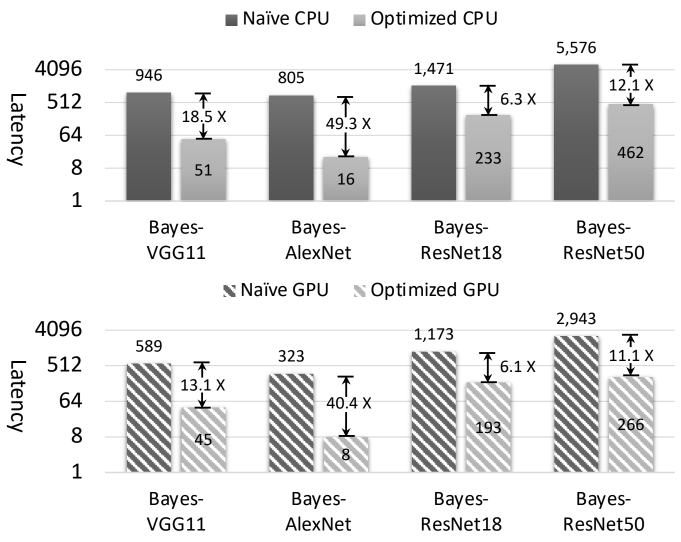


Fig. 14. Effect of our framework on CPU and GPU.

TABLE 3
Performance comparison of FPGA design versus GPU implementation. S: sample skipping, L: layer skipping, C: channel skipping.

	GPU				Our Work			
Platform	GeForce RTX 2080 Ti				Intel Arria 10 GX1150			
Frequency	1.545 GHz				220 MHz			
Technology	12 nm				20 nm			
Acceleration Library	CuDNN, PyTorch 1.9.0				-			
Power (W)	236				45			
Model	*Bayes-VGG11*		*Bayes-ResNet50*		*Bayes-VGG11*		*Bayes-ResNet50*	
Version (ms)	Naive	Opt (S&L)	Naive	Opt (S&L)	Opt (S&L)	Opt (S&L&C)	Opt (S&L)	Opt (S&L&C)
Latency (ms)	591.1	45.132	2966	267.23	9.45	5.9	43.73	37.70
Energy Eff. (J/Frame)	138.9	10.61	700.1	63.07	0.42	0.27	1.97	1.70

- The hardware optimization that carefully chooses the parallelism strategies for different BayesNNs.
- Multiple consecutive layers are performed in a fused manner based on an integrated hardware engine, which significantly decreases the memory traffic.

5.5 Comparison with the Existing Design

To demonstrate the benefits of our hardware architecture and optimization framework as a whole, we compared our work against the existing designs in Table 4. Both Cai *et al.* [42] and Awano *et al.* [41] only accelerated Bayes-FC that only consisted of FC layers. The hardware designed by Wan *et al.* [12] was not able to accelerate BayesNNs with residual connections. Also, Fan *et al.* [40] only evaluated their accelerator with small BayesNNs such as Bayes-VGG11 and Bayes-ResNet18 they and did not support layer-wise dropout rate. Therefore, our accelerator is more versatile than the existing designs. As these designs were evaluated on different BayesNNs, it was unfair to compare them in terms of the latency, hence, we focused on throughput, energy efficiency by GOP/s per Watt. The total number of operations was obtained by accumulating the computation required by every layer and MC sample. The original input image size was 224×224 with 3 channels. We quote the hardware performance from the original papers for [42], [41] and [40]. In [40], as the authors presented several designs with different optimization objectives, we choose the one with the highest hardware performance. Compared with [40], we can achieve nearly 1.6 and 1.2 times speedup on *Bayes-VGG11* and *Bayes-ResNet18* respectively. There are two reasons for the improvement of throughput compared with [40]: *a)* our proposed hardware was able to intelligently skip redundant channels; *b)* the proposed framework systematically explored three types of structured sparsity. In [12], since the authors only reported the normalized speedup without mentioning the real processing speed in the original paper, we were not able to compare against it directly. Since they also focused on exploiting sparsity to accelerate BayesNNs, we compared with [12] in terms of the speedup brought by exploiting sparsity. As it can be observed, [12] achieved 2.4~3.1$\times$ speedup, while our work improved the performance by 7.8~27.9 times. We achieved better performance than [12] by exploiting three categories of structured sparsity with a dedicated hardware design.

TABLE 4
Performance comparison of our final FPGA designs with the related work.

	ASPLOS'18 [42]	DATE'20 [41]	DAC'21 [40]		Micro'20 [12]		Our Work		
Platform	Altera Cyclone V	Zynq XC7Z020	Arria 10 GX1150		Virtex-7 VC709		Arria 10 GX1150		
Frequency (MHz)	213	200	225		100		220		
Technology	28 nm	28 nm	20 nm		28 nm		20 nm		
Used DSPs	342	220	1518		3600		1606		
Power (W)	6.11	2.76	45.00		-		43.6		
Model	*Bayes-FC*	*Bayes-FC*	*Bayes-VGG11*	*Bayes-ResNet18*	*Bayes-LeNet5*	*Bayes-GoogLeNet*	*Bayes-VGG11*	*Bayes-ResNet18*	*Bayes-ResNet50*
Throughput (GOP/s)	59.6	24.22	534	1590	-	-	854.4	1812.6	1489.8
Speedup by Exploiting Sparsity	-	-	-	-	2.4 ×	3.1×	27.9×	7.8×	12.8×
Energy Efficiency (GOP/s/W)	9.75	8.77	11.9	33.3	-	-	19.6	41.57	34.2

6 Related Work

6.1 Exploiting Sparsity in NN Accelerators

Various hardware architectures have been proposed to accelerate sparse convolution (SpCONV) and sparse general matrix-matrix multiplication (SpGEMM). Both *Sparten* [21] and *Extensor* [22] proposed dedicated hardware architectures to accelerate inner-product-based SpCONV. Targeting on accelerating SpGEMM [23], Zhang *et al.* [24] also proposed outer-product-based methods to achieve a high hardware performance. Wang *et al.* [26] leveraged an outer-product-based approach to accelerate both SpGEMM and SpCONV. Apart from accelerating standard convolutional NNs (CNNs), various accelerators have been proposed to accelerate graph neural networks (GNNs) by exploiting their sparsity [43], [44], [45]. However, these accelerators for CNNs and GNNs mainly focused on exploiting unstructured sparsity: the irregular zeros in both activation and weights introduced by pruning or ReLU activation, which often require complicated and costly hardware design to achieve performance benefits. As unstructured sparsity only occupies a small portion of computation in Bayesian convolutional NNs (BayesCNNs) [12] compared with structured sparsity, the speedup they can achieve is limited.

Along with hardware architecture support, various pruning techniques [46] have been proposed to compress the standard NNs, which introduced extensive levels of unstructured sparsity (point-wise pruning [13]) or structured sparsity (channel pruning [47] and filter pruning [48]). Orthogonal to the sparsity in standard NNs, the structured sparsity utilized in our paper is introduced by MCD and subsequent MC sampling in BayesNNs. Therefore, the traditional sparsity in standard NNs is orthogonal to the presented structured sparsity, so both can be used together to further improve the hardware performance of BayesNNs.

6.2 Accelerators for BayesNNs

Although various accelerators have been proposed to accelerate NNs [49], accelerating BayesNNs has not attained a similar level of research interests. *VIBNN* [42] was the first work to accelerate FC Bayesian NNs. *BYNQNet* [41] improved the hardware performance by exploring the sampling-free Bayesian NNs. However, these works only considered the accuracy without including uncertainty estimation performance during their evaluation. Additionally, their designs only support BayesNNs consisting only of FCs, which prevents them from generalising to modern convolutional architectures [14]. Fan *et al.* [40] proposed an FPGA-based hardware architecture for MCD-based BayesCNNs. In parallel to this work, Rock *et al.* [34], while extending [1], discuss algorithmic optimizations consisting of exploiting layer and channel sparsity and different dropout rates during training and evaluation to achieve gains in hardware performance. *Fast-BCNN* [12] accelerated BayesCNNs by skipping the zeros produced by the element-wise ReLU activation. Given that other activation functions such as leaky ReLU have been widely adopted, the generality and efficiency of their design is again limited.

In contrast to the previous work, this paper systematically and jointly exploits three categories of structured sparsity per layer, to improve the hardware performance of BayesNNs without sacrificing their algorithmic performance through the proposed optimization framework. Our proposed hardware architecture is designed to intelligently skip redundant layers, channels and samples to exploit structured sparsity in BayesNNs.

7 Conclusion

This paper proposes to accelerate Bayesian NNs by exploiting the structured sparsity from both algorithmic and hardware perspectives. We observe and categorise structured sparsity in BayesNNs as channel sparsity, layer sparsity and sample sparsity. A novel hardware architecture is proposed to skip the redundant computation introduced by three types of structured sparsity. As higher sparsity leads to better hardware performance on our design, we propose a framework to automatically explore structured sparsity in Bayesian NNs without sacrificing algorithmic performance. Extensive experiments on four BayesNNs and datasets demonstrated that our design, together with the optimization framework, can achieve up to 39.3 times higher energy efficiency than the GPU implementation and up to 1.6 times speedup compared with the state-of-the-art designs.

References

[1] T. Azevedo, R. de Jong, M. Mattina, and P. Maji, "Stochastic-YOLO: Efficient probabilistic object detection under dataset shifts," *arXiv preprint arXiv:2009.02967*, 2020.

[2] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the 2018 European Conference on Computer Vision (ECCV)*. Springer, 2018, pp. 801–818.

[3] M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. Fieguth, X. Cao, A. Khosravi, U. R. Acharya *et al.*, "A review of uncertainty quantification in deep learning: Techniques, applications and challenges," *Information Fusion*, 2021.

[4] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of the 2016 International Conference on Machine Learning (ICML)*. PMLR.org, 2016, pp. 1050–1059.

[5] A. O. Aseeri, "Uncertainty-aware deep learning-based cardiac arrhythmias classification model of electrocardiogram signals," *Computers*, vol. 10, no. 6, p. 82, 2021.

[6] D. Feng, L. Rosenbaum, and K. Dietmayer, "Towards safe autonomous driving: Capture uncertainty in the deep neural network for lidar 3d vehicle detection," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 3266–3273.

[7] M. Wen and E. B. Tadmor, "Uncertainty quantification in molecular simulations with dropout neural network potentials," *npj Computational Materials*, vol. 6, no. 1, pp. 1–10, 2020.

[8] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," in *Proceedings of the 2015 32nd International Conference on Machine Learning (ICML)*. PMLR.org, 2015, p. 1613–1622.

[9] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient Langevin dynamics," in *Proceedings of the 2011 28th International Conference on Machine Learning (ICML)*. PMLR.org, 2011, pp. 681–688.

[10] Z. Ghahramani, "Probabilistic machine learning and artificial intelligence," *Nature*, vol. 521, no. 7553, pp. 452–459, 2015.

[11] J. van der Westhuizen and J. Lasenby, "Bayesian LSTMs in medicine," *arXiv preprint arXiv:1706.01242*, 2017.

[12] Q. Wan and X. Fu, "Fast-BCNN: Massive neuron skipping in Bayesian convolutional neural networks," in *Proceedings of the 2020 Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 229–240.

[13] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 770–778.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[16] Z. Zhang, A. V. Dalca, and M. R. Sabuncu, "Confidence calibration for convolutional neural networks using structured dropout," *arXiv preprint arXiv:1906.09551*, 2019.

[17] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," *arXiv preprint arXiv:1511.02680*, 2015.

[18] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, "Understanding deep neural networks with rectified linear units," *arXiv preprint arXiv:1611.01491*, 2016.

[19] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *arXiv preprint arXiv:1506.02626*, 2015.

[20] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 27–40, 2017.

[21] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. Vijaykumar, "SparTen: A sparse tensor accelerator for convolutional neural networks," in *Proceedings of the 2019 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2019, pp. 151–165.

[22] K. Hegde, H. Asghari-Moghaddam, M. Pellauer, N. Crago, A. Jaleel, E. Solomonik, J. Emer, and C. W. Fletcher, "Extensor: An accelerator for sparse tensor algebra," in *Proceedings of the 2019 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2019, pp. 319–333.

[23] S. Pal, J. Beaumont, D.-H. Park, A. Amarnath, S. Feng, C. Chakrabarti, H.-S. Kim, D. Blaauw, T. Mudge, and R. Dreslinski, "Outerspace: An outer product based sparse matrix multiplication accelerator," in *Proceedings of the 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 724–736.

[24] Z. Zhang, H. Wang, S. Han, and W. J. Dally, "Sparch: Efficient architecture for sparse matrix multiplication," in *Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 261–274.

[25] N. Srivastava, H. Jin, J. Liu, D. Albonesi, and Z. Zhang, "Matraptor: A sparse-sparse matrix multiplication accelerator based on row-wise product," in *Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 766–780.

[26] Y. Wang, C. Zhang, Z. Xie, C. Guo, Y. Liu, and J. Leng, "Dual-side sparse tensor core," *Proceedings of the 2021 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2021.

[27] A. L. Maas, A. Y. Hannun, A. Y. Ng *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of the 2013 International Conference on Machine Learning (ICML)*, vol. 30. PMLR.org, 2013, p. 3.

[28] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[29] A. Kristiadi, M. Hein, and P. Hennig, "Being Bayesian, even just a bit, fixes overconfidence in ReLU networks," *arXiv preprint arXiv:2002.10118*, 2020.

[30] Y. LeCun, "The MNIST database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[31] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011. [Online]. Available: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf

[32] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," *arXiv preprint arXiv:1706.04599*, 2017.

[33] S. Liu, H. Fan, M. Ferianc, X. Niu, H. Shi, and W. Luk, "Toward full-stack acceleration of deep convolutional neural networks on FPGAs," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14, 2021.

[34] J. Rock, T. Azevedo, R. de Jong, D. Ruiz-Muñoz, and P. Maji, "On efficient uncertainty estimation for resource-constrained mobile applications," *arXiv preprint arXiv:2111.09838*, 2021.

[35] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2704–2713.

[36] M. Ferianc, P. Maji, M. Mattina, and M. Rodrigues, "On the effects of quantisation on model uncertainty in bayesian neural networks," in *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*. PMLR, 2021, pp. 929–938.

[37] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Proceedings of the 2019 Advances in neural information processing systems (NeurIPS)*, vol. 32, pp. 8026–8037, 2019.

[39] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou *et al.*, "Mlperf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 446–459.

[40] H. Fan, M. Ferianc, M. Rodrigues, H. Zhou, X. Niu, and W. Luk, "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.

[41] H. Awano and M. Hashimoto, "Bynqnet: Bayesian neural network with quadratic activations for sampling-free uncertainty estimation on FPGA," in *Proceedings of the 2020 Design, Automation & Test*

in Europe Conference & Exhibition (DATE). IEEE, 2020, pp. 1402–1407.

[42] R. Cai, A. Ren, N. Liu, C. Ding, L. Wang, X. Qian, M. Pedram, and Y. Wang, "Vibnn: Hardware acceleration of Bayesian neural networks," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 476–488, 2018.

[43] H. Zeng and V. Prasanna, "GraphACT: Accelerating GCN training on CPU-FPGA heterogeneous platforms," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 2020, p. 255–265.

[44] T. Geng, A. Li, R. Shi, C. Wu, T. Wang, Y. Li, P. Haghi, A. Tumeo, S. Che, S. Reinhardt *et al.*, "AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing," in *Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 922–936.

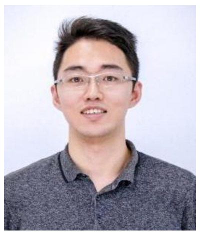
[45] S. Liang, Y. Wang, C. Liu, L. He, H. LI, D. Xu, and X. Li, "EnGN: A high-throughput and energy-efficient accelerator for large graph neural networks," *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1511–1525, 2021.

[46] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," *arXiv preprint arXiv:2102.00554*, 2021.

[47] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 1389–1397.

[48] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 4340–4349.

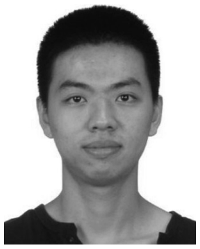
[49] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey of machine learning accelerators," in *Proceedings of the 2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2020, pp. 1–12.



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 designing domain-specific hardware architecture for AI algorithms and applying ML algorithm on system-level and chip-level optimizations.

Martin Ferianc is a PhD candidate in the Department of Electronic and Electrical Engineering at University College London. His research interests include Neural architecture search, Bayesian neural network, Deep Learning and Hardware acceleration of neural networks. Martin has obtained an MEng in Electronic and Information Engineering from Imperial College London.

Zhiqiang Que is a research assistant pursuing his Ph.D. degree in the department of Computing, Imperial College London, UK. He received his B.S in Microelectronics and M.S in CS from Shanghai Jiao Tong University in 2008 and 2011 respectively. From 2011 to 2016, he worked on microachitecture design and verification of ARM CPUs with the Marvell semiconductor Ltd., Shanghai. His research interests include computer architectures, embedded systems, high-performance computing and computer-aided design (CAD) tools for hardware design optimization.



Xinyu Niu is the Co-Founder and CEO of Corerain Technologies in Shenzhen, China. He received the B.Sc. Degree from Fudan University, Shanghai, China, and the M.Sc. and Ph.D. degrees in computing science from Imperial College London, London, U.K. His current research interests include developing applications and tools for reconfigurable computing that involves runtime reconfiguration.

Miguel R. D. Rodrigues (Senior Member, IEEE) received the Licenciatura degree in electrical and computer engineering from the University of Porto, Porto, Portugal, and the Ph.D. degree in electronic and electrical engineering from the University College London (UCL), London, U.K. He is currently a Professor of Information Theory and Processing, UCL, and a Turing Fellow with the Alan Turing Institute - the UK National Institute of Data Science and Artificial Intelligence. His research lies in the general areas of information theory, information processing, and machine learning. His work has led to more than 200 articles in leading journals and conferences in the field, a book on Information-Theoretic Methods in Data Science (Cambridge Univ. Press), and the IEEE Communications and Information Theory Societies Joint Paper Award 2011.

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.