FPGA-based Acceleration for Bayesian Convolutional Neural Networks

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

***Abstract*—Neural networks (NNs) have demonstrated their potential in a variety of domains ranging from computer vision to natural language processing. Among various NNs, two-dimensional (2D) and three-dimensional (3D) convolutional neural networks (CNNs) have been widely adopted for a broad spectrum of applications such as image classification and video recognition, due to their excellent capabilities in extracting 2D and 3D features. However, standard 2D and 3D CNNs are not able to capture their model uncertainty which is crucial for many safety-critical applications including healthcare and autonomous driving. In contrast, Bayesian convolutional neural networks (BayesCNNs), as a variant of CNNs, have demonstrated their ability to express uncertainty in their prediction via a mathematical grounding. Nevertheless, BayesCNNs have not been widely used in industrial practice due to their compute requirements stemming from sampling and subsequent forward passes through the whole network multiple times. As a result, these requirements significantly increase the amount of computation and memory consumption in comparison to standard CNNs. This paper proposes a novel FPGA-based hardware architecture to accelerate both 2D and 3D BayesCNNs based on Monte Carlo Dropout. Compared with other state-of-the-art accelerators for BayesCNNs, the proposed design can achieve up to 4 times higher energy efficiency and 9 times better compute efficiency. An automatic framework capable of supporting partial Bayesian inference is proposed to explore the trade-off between algorithm and hardware performance. Extensive experiments are conducted to demonstrate that our framework can effectively find the optimal implementations in the design space.**

***Index Terms*—Bayesian convolutional neural network (BayesCNN), Three-dimensional convolutional neural network (3D CNN), Field-programmable gate array (FPGA), Deep learning**

I. Introduction

The past decade has witnessed a great success of neural networks (NNs) in a wide range of artificial intelligence (AI) tasks [1], [2]. Among various NNs, two-dimensional (2D) convolutional neural networks (CNNs) have demonstrated their excellent algorithm performance primarily in 2D computer vision (CV) applications, such as in semantic segmentation [3], [4] or object detection [5], due to their ability to extract 2D patterns. To incorporate temporal information into analysis and prediction, three-dimensional (3D) CNNs have been proposed [6], which are more suitable for 3D CV applications such as human action recognition [7] or video segmentation [8]. Although both 2D and 3D CNNs have become popular in various CV applications, these models are not able to express their model uncertainty which is crucial for a variety of safety-critical applications such as in healthcare [9] or autonomous vehicles [10].

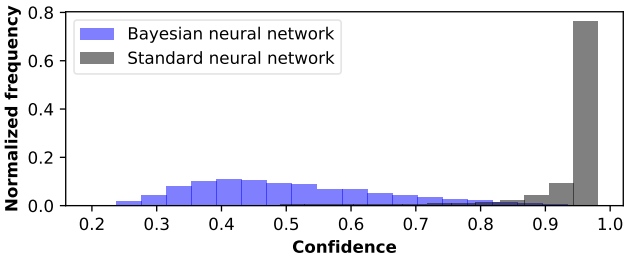


Fig. 1. Comparison of prediction confidence histograms for random noise input between Bayesian neural network and a standard neural network. Bayesian neural network correctly shows higher uncertainty, and its confidence histogram is wider and with a lower normalised frequency than a standard neural network.

Bayesian convolutional neural networks (BayesCNNs) [11], including both 2D BayesCNNs [12] and 3D BayesCNNs [13], represent a variant of CNNs that are able to treat their uncertainty by modelling their weights as probability distributions. A comparison between 2D BayesCNN and a standard 2D CNN in an image classification task is presented in Figure 1. Given a completely irrelevant input, 2D BayesCNN demonstrates low confidence and thus high uncertainty, while the standard 2D CNN is overconfident in its incorrect prediction. Therefore, through the use of BayesCNNs, practitioners are able to explicitly capture special cases [12] and they have become relevant to applications where the notion of uncertainty estimation is essential, such as in medicine [4].

Nevertheless, the algorithm complexity of BayesCNNs puts a large burden on their real-world hardware performance. Due to the high dimensionality of modern BayesCNNs, it is intractable to analytically compute their predictive uncertainty. Instead, the predictive distribution can be approximated by Monte Carlo (MC) sampling that requires repeated sampling of the weight distributions, and then the input data are run through the BayesCNNs multiple times, which can

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, the National Natural Science Foundation of China (No. 62001165), Hunan Provincial Natural Science Foundation of China (No. 2021JJ40357), Changsha Municipal Natural Science Foundation (No. kq2014079) and in part by the funds from Corerain, Maxeler, Intel, Xilinx and SGIIT. 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.

S. Liu is with the School of Physics and Electronics, Hunan Normal University, Changsha 410081, China.

X. Niu is with Corerain Technologies Ltd., Shenzhen, China.

† Equal Contribution.

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

prevent real-time applications. For instance, Bayesian *ResNet-18* requires nearly 132 seconds to predict the outcome with respect to only a single input on an Intel Core i9-9900K CPU, which cannot meet the requirements of practical real-time applications [14]. This problem is further exacerbated while considering 3D BayesCNNs, as they are more compute-intensive and memory-intensive than 2D BayesCNNs [15]. Therefore, there is a great demand for specific hardware designs for accelerating BayesCNNs.

Although several hardware accelerators and algorithm approximation techniques [16]–[19] have been proposed to improve the hardware performance of BayesCNNs, there are several drawbacks in these approaches:

- The implementation needs of both a compute engine and weight samplers makes the hardware design resource and memory-demanding, which degrades the hardware performance of BayesCNNs.
- Current accelerators can only support 2D BayesCNNs. It is especially challenging to accelerate Bayesian 3D CNNs as they require more memory and computation.
- To obtain the uncertainty estimation, these accelerators repeatedly perform S forward passes through the whole BayesCNN without considering whether it is actually necessary from the algorithm perspective, which makes them S times slower than standard CNNs.

To address these challenges, we propose an field-programmable gate array (FPGA)-based hardware design to accelerate BayesCNNs based on Monte Carlo Dropout (MCD) [12], [20]. The proposed accelerator is versatile to run both 2D and 3D BayesCNNs with the support of fine-grained parallelism. To improve the hardware performance, we consider partial BayesCNNs to decrease the amount of computation required. An intermediate-layer caching (IC) hardware implementation is also proposed to reduce the on-chip and off-chip memory traffic. To explore the trade-off between algorithm and hardware performance, an automatic framework is proposed to find the suitable hardware configurations and algorithm parameters given users' hardware constraints and algorithm requirements. We choose FPGAs as our hardware platform because of its better flexibility over ASICs, which we utilize in our hardware optimizations for different BayesCNNs, and higher energy efficiency than GPUs [21]–[27].

In summary, our contributions include:

- A novel hardware architecture to accelerate Monte Carlo Dropout-based 2D and 3D Bayesian convolutional neural networks (BayesCNNs). A new intermediate-layer caching technique also helps the proposed design to achieve high performance and efficiency (Section III and IV).
- An automatic framework for both 2D and 3D BayesCNNs to explore algorithm-hardware performance trade-off provided by partial Bayesian inference under user-defined constraints (Section V).
- A comprehensive evaluation of algorithm and hardware performance on different datasets with respect to various state-of-the-art Bayesian 2D and 3D CNNs (Section VI), which demonstrates the versatility and effectiveness of our hardware architecture and automatic framework.

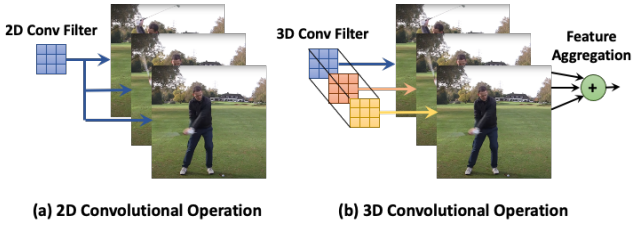


Fig. 2. 2D and 3D convolutional operations. The channel dimension of feature maps is not shown for simplicity.

This work extends our conference publication [14]. The extended material includes: *1)* an improved hardware architecture to accelerate both 2D and 3D BayesCNNs; *2)* an extended automatic framework to optimize the algorithm and hardware performance for both 2D and 3D BayesCNNs; *3)* an extensive evaluation of a wide range of BayesCNNs targeting various 2D and 3D image datasets.

II. Background

A. 2D and 3D Convolutional neural networks

In general, 2D or 3D CNNs are constructed by sequential layering of 2D or 3D convolution and pooling operations, which gradually refines the input into a prediction [28]. 2D CNNs have been widely deployed in various 2D computer vision (CV) applications such as image segmentation [4], image classification [29] or object detection [30].

To efficiently extract 2D spatial features, 2D CNNs adopt the 2D convolutional operation to process and analyse images, which is illustrated in Figure 2(a). In this example, a 2D convolutional filter (2D Conv Filter) with $K_H = K_W = 3$ is applied to all the pixels in a sliding-window fashion in an input channel to extract relevant features. The notation used in this paper is summarised in Table I.

However, when considering 3D data with an additional temporal dimension, such as a video clip, the 2D convolution is

TABLE I
Parameter notation used in this paper.

Parameter	Description
H	The height of input feature map
W	The width of input feature map
K_H	The kernel height
K_W	The kernel width
K_L	The kernel length along temporal dimension
C	The number of channels
F	The number of filters
L	The length along temporal dimension
PV	The data parallelism level
PC	The channel parallelism level
PF	The filter parallelism level
N	Number of layers or network depth
S	Number of Monte Carlo samples
B	Number of Bayesian layers from the end
N_{lfsr}	Number of LFSRs to implement Bernoulli sampler
p	Dropout probability
EFF_{io}	IO communication efficiency
$FREQ_{io}$	IO clock frequency
$FREQ_{pl}$	Clock frequency of programmable logic (PL)
D_{FIFO}	The depth of FIFO
BW	Bandwidth between on-chip and off-chip memory
V	Data bit-width
DSP	DSP consumption
MEM	Memory consumption

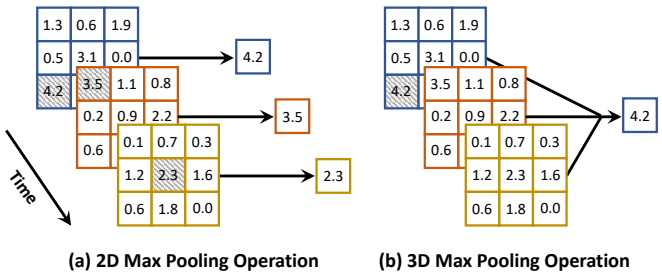


Fig. 3. 2D and 3D Max Pooling Operations.

no longer suitable. In 2D CNNs, 2D convolution would use the same 2D filter for different temporal frames, and thus it would not be able to capture the temporal information. To address this issue, 3D CNNs were proposed [31] with the ability of incorporating the third-dimensional information. Due to their capability to factor in the third-dimensional information in their prediction, 3D CNNs have been widely adopted in 3D CV tasks such as human action recognition [30], [31] and video segmentation [32]. 3D CNNs adopt 3D convolution that applies different convolutional filters for different consecutive frames along the temporal dimension. The results from different convolutions-frames are then aggregated together to generate output feature maps. Hence, the 3D CNN is able to capture and preserve the information existing in the third-temporal dimension. An example of 3D convolution with a 3D convolutional filter (3D Conv Filter) ($K_H = K_W = K_L = 3$) is illustrated in Figure 2(b). The 3D convolutional filter is formed as a cube and each frame, given a video input, is processed by a different filter with kernel size $3 \times 3 \times 3$.

Apart from convolutional operations, 2D and 3D CNNs adopt different pooling operations, namely 2D and 3D pooling to distillate the retrieved information from previous convolutions. An example of 2D and 3D maximum pooling (Max Pooling) is presented in Figure 3(a,b). In 2D CNNs, the 2D pooling is applied in each frame separately, and thus it can only distillate information to reduce the input size in the spatial dimension. Whereas, 3D CNNs perform 3D pooling in both spatial and temporal dimensions, and therefore 3D pooling operations are able to consider all three dimensions by reducing the input size in both temporal and spatial dimensions.

B. Bayesian Convolutional Neural Networks

Bayesian inference [11] aims to make the previously discussed 2D or 3D CNNs more robust to overfitting and enable them to quantify their model uncertainty [33]. Bayesian inference does this through learning a distribution over the weights of the NN, e.g. when considering the previously described Conv filters, instead of pointwise estimates, adopt a Bayesian NN or a BayesCNN. The learning of the distribution $p(\boldsymbol{w}|\mathcal{D})$ for the weights $\boldsymbol{w}$ with respect to some observed data $\mathcal{D}$ is hence done through the Bayes rule. The Bayes rule combines the belief about the noise in the data in the form of the likelihood $p(\mathcal{D}|\boldsymbol{w})$ and the prior distribution over weights $p(\boldsymbol{w})$, such that $p(\boldsymbol{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})}{p(\mathcal{D})}$. $p(\boldsymbol{w}|\mathcal{D})$ is the target posterior distribution over the weights $\boldsymbol{w}$ that we wish to learn and use to make predictions about previously unobserved data.

Nevertheless, due to the high dimensionality of modern Bayesian NNs it is intractable to analytically compute the posterior distribution $p(\boldsymbol{w}|\mathcal{D})$ and it needs to be approximated with respect to a parametrizable variational distribution $q(\boldsymbol{w}|\boldsymbol{\theta}, \mathcal{D})$ and some learnable parameters $\boldsymbol{\theta}$. The resultant distribution $q(.)$ can then be used to make predictions for previously unseen data $\mathcal{D}^*$ through an integral $p(\mathcal{D}^*) = \int p(\mathcal{D}^*|\boldsymbol{w})q(\boldsymbol{w}|\boldsymbol{\theta}, \mathcal{D})d\boldsymbol{w}$. This integral is again intractable due to the nonexistent closed-form analytical solution of the variational posterior and it needs to be approximated through MC sampling with S samples as shown in (1):

$$p(\mathcal{D}^*) = \frac{1}{S}\sum_{s=1}^{S} p(\mathcal{D}^*|\boldsymbol{w}_s); \quad \boldsymbol{w}_s \sim q(\boldsymbol{w}|\boldsymbol{\theta}, \mathcal{D}) \tag{1}$$

Given $p(\mathcal{D}^*)$, it is then possible to quantify the previously discussed uncertainty. The required repeated runs with respect to the variational posterior and sampling demand efficient hardware processing to reduce the compute cost of the forward pass through the Bayesian NN S times. To target this challenge, this work proposes an efficient FPGA-based accelerator for Bayesian NNs as well as an accompanying automatic framework for optimization.

1) Monte Carlo Dropout (MCD): Monte Carlo Dropout (MCD) approaches the learning of the variational distribution over weights $q(\boldsymbol{w}|\boldsymbol{\theta}, \mathcal{D})$ [12], [34] as casting dropout [35] with weight decay or L2 regularisation as Bayesian inference. The concept of dropout lays in randomly dropping out connections in a NN in order to achieve more robust feature detection and independence between neighboring units. Hence, MCD can be described as applying a random filter-wise mask $\boldsymbol{M}_i \in \mathbb{R}^{F_i}$, to the output feature maps $\boldsymbol{Y}_i$ of layer i with F_i dimensional filters [20]. The mask $\boldsymbol{M}_i$ follows a Bernoulli distribution $p(\boldsymbol{M}_i|p_i)$ which generates binary random variables (0 or 1) with the probability $p_i \in (0, 1)$. To get the final output $\boldsymbol{O}_i$ for layer i under MCD, the computation can be formulated as shown in (2):

$$\boldsymbol{O}_i = \boldsymbol{Y}_i \boldsymbol{M}_i \tag{2}$$

The uncertainty estimation and prediction are thus obtained by running the same input through the BayesCNNs S times, each time with a different set of sampled masks $\boldsymbol{M}$ which translate into sampling $q(\boldsymbol{w}_i|\boldsymbol{\theta}_i, \mathcal{D})$ for each layer i where MCD is applied, and averaging the outputs as shown in (1). The $\boldsymbol{\theta}_i$ are the learnt variational parameters, e.g. the Conv filters. Unlike the dropout used in standard NNs which is only enabled during training, MCD applies the dropout during both training and evaluation. The works [4], [12], [20], [36] demonstrate that MCD can achieve a high quality of uncertainty estimation.

2) Partial Bayesian Inference: A completely Bayesian NN should be inferred such that every layer's parameters are modelled as distributions instead of pointwise estimates, which translates into applying MCD to every layer [12]. However, the authors in [10], [36]–[38] have demonstrated theoretically and empirically that combining Bayesian and non-Bayesian layers in the same network, and thus making the network partially Bayesian, can bring algorithm benefits such as improved uncertainty estimation and accuracy. Assuming there is an N-layer NN, in this work we assume ordered partial Bayesian NN

by being Bayesian in the last B layers ($B \leq N$) layers which makes the first $N - B$ layers behave as a feature extractor for the Bayesian remainder. Partially applied dropout then enables trade-off between algorithm and hardware performance and uncertainty estimation [36]. In this work, we explore this trade-off by proposing a framework for determining the positioning of MCD at different parts of the NN which results in a partial Bayesian NN.

C. Related Work

1) FPGA-based CNN Accelerator: Various FPGA-based accelerators have been proposed to accelerate standard 2D CNNs with high energy, compute, memory and resource consumption efficiency. By thoroughly exploring the hardware design space and utilizing the roofline model [39] Zhang *et al.* [21] optimized an FPGA-based accelerator for 2D CNNs. Ma *et al.* [22] further improved the hardware performance of 2D CNNs on FPGAs by studying and exploiting the convolution's looping. Yu *et al.* [23] proposed an end-to-end compiler called *DNNVM* that leverages heterogeneous optimizations to accelerate 2D CNNs on FPGAs. However, the support of 3D CNNs on these designs was not established and optimized.

Fan *et al.* [15], [40] proposed an FPGA-based accelerator for 3D CNNs with an optimized computational pattern to decrease the memory traffic between the off-chip and on-chip memory. By utilizing different spatial and temporal tiling strategies, Kartik *et al.* [41] proposed *Morph* to accelerate 3D CNNs with high energy efficiency. While both [15], [41] focused on accelerating 3D CNNs, Shen *et al.* [42] were the first to accelerate both 2D and 3D CNNs on a uniform template-based architecture using a Winograd algorithm [43]. However, the use of the Winograd algorithm requires extra logic resources to implement the transformation of input and output matrices. Liu *et al.* [44] proposed an uniform architecture based on 2D multiply-accumulate (MAC) array. Their design uses low bit-width fixed-point arithmetic and hence the algorithm accuracy cannot be guaranteed.

At the same time, a significant research effort has been invested into studying compression techniques for acceleration, such as quantization [45]–[48], pruning [49], [50] and neural architecture adaptation towards hardware-efficient deployment [51], [52]. A comprehensive literature review of compression techniques for custom hardware was summarized in [24]. However, these techniques and designs mainly focus on accelerating standard CNNs, without considering the support for 2D or 3D BayesCNNs.

2) Acceleration for BayesCNNs: Compared with standard CNNs, BayesCNNs are more compute and memory-intensive as discussed in Section II-B. They require the MC sampling which demands repeated random number generation and multiple feed-forward passes to obtain the predictive distribution and hence quantification of uncertainty. Cai *et al.* [16] proposed a hardware design named *VIBNN* to accelerate Bayesian NNs on an FPGA. However, *VIBNN* can only support linear feed-forward layers, which limits its generality to solving real-world problems. Myojin *et al.* [17] proposed an FPGA-based accelerator with multiple computational engines that run multiple samples in parallel. However, their design can only support binarised Bayesian NNs. An efficient inference algorithm for Bayesian NNs named *BYNQNet* with a corresponding FPGA-based implementation was proposed in [18]. The work employs quadratic nonlinear activation functions to simplify the hardware design for sampling-free Bayesian NNs. However, the restriction placed on the nonlinear activation function limits their applicability to modern architectures. Wan *et al.* [19] proposed an FPGA-based hardware accelerator that intelligently skips the redundant computations in 2D BayesCNNs. Nevertheless, its support for Bayesian 3D CNNs is unknown and several algorithm metrics such as the quality of uncertainty prediction and confidence were not evaluated and explored. Jia *et al.* [53] proposed the feature decomposition and memorization techniques to reduce the computation required by 2D BayesCNNs. Nevertheless, their optimization is only performed on the software level.

Different from the previous work, our paper proposes a unified hardware architecture to accelerate both Bayesian 2D and 3D CNNs based on MCD. To further improve the hardware performance, we overlap the Bernoulli sampling with the computation, and propose a hardware-efficient intermediate layer caching implementation to reduce the unnecessary memory traffic and computation. By exploiting partial Bayesian inference, an automatic tool is introduced to explore algorithm-hardware design space.

III. Hardware Development

A. Hardware Design

1) Design Overview: An overview of our proposed FPGA-based accelerator is presented in Figure 4. It mainly consists of a neural network engine (NNE) and a Bernoulli sampler along with an interface to interact with the off-chip memory. As the channel size, filter size and the size of feature maps vary among different BayesCNNs, the NNE is designed to support three categories of configurable parallelism, including channel parallelism (PC), vector parallelism (PV) and filter parallelism (PF) to meet different algorithm needs.

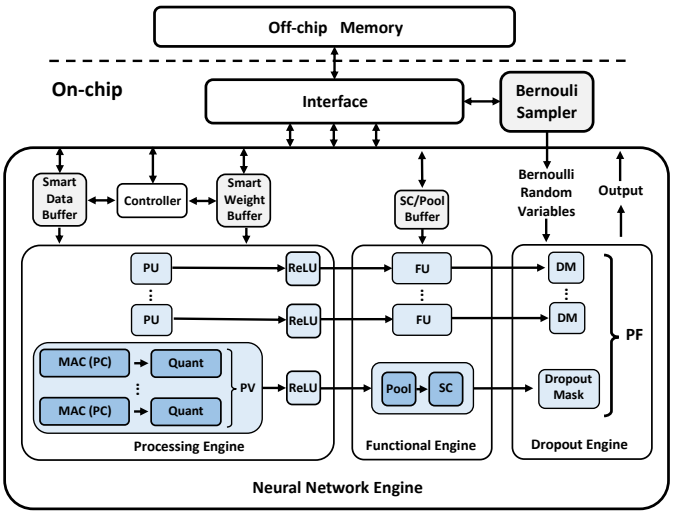


Fig. 4. Overview of the FPGA-based accelerator.

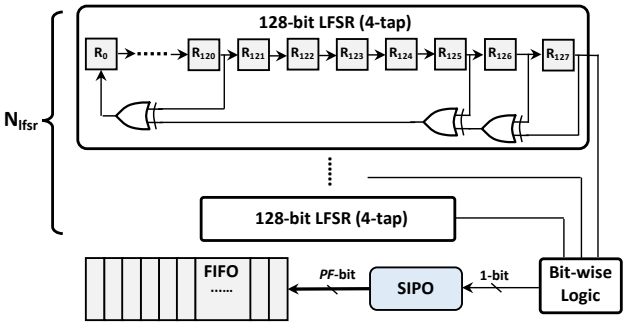


Fig. 5. Hardware architecture of the Bernoulli sampler.

2) Neural Network Engine (NNE): We design the NNE to perform 2D or 3D convolution and its following functional layers such as rectified linear unit (ReLU) and pooling at a given time. The NNE is composed of a processing engine (PE), a functional engine (FE), a dropout engine (DE) and different memory buffers to cache inputs, weights and intermediate results. In PE, there are PC processing units (PUs) with each PU followed by a ReLU unit which can be optionally bypassed. The PU is where we perform the 2D or 3D convolutional operation. At the beginning of the PU is a MAC unit with PC multipliers followed by a log_2PC-level adder tree and an accumulator. As our accelerator adopts the 8-bit linear quantization [54] to improve the overall hardware performance, a quantization (Quant) module is designed after the MAC unit to map the accumulated 32-bit results back to 8-bit integers for the use in the next layer. The functional engine is designed to perform 2D or 3D pooling and shortcut (SC) addition. Since the 3D pooling needs compute results from adjacent frames and the SC addition requires to perform addition between the outputs and the input, a memory buffer is designed to cache the necessary data required by both operations. To perform dropout, the DE receives the Bernoulli random variables (1 or 0) from the Bernoulli sampler and creates a mask to randomly drop intermediate output filters.

As there are limited on-chip memory resources on FPGAs, especially considering FPGAs for edge deployment, it is impossible to cache all intermediate results of multiple convolutional layers in a deep BayesCNNs on-chip. To improve the generality of our accelerator, we only deploy one NNE on-chip and run the whole network layer-by-layer using the same NNE. In this manner, outputs generated from the DE will be transferred back to the off-chip memory from the on-chip memory. At the same time, the inputs of the next convolutional layer need to be loaded from off-chip memory into the on-chip memory. To decrease the memory transfer time between on-chip and off-chip memory, double buffer technique is used for both input buffer and weight buffer design to hide the transfer time into the computation time [55].

3) Bernoulli Sampler: To perform MCD, it is required to generate random 1s and 0s at runtime to randomly dropout some filters in the output feature map as illustrated in (2). The probability of 1s and 0s is specified by the dropout rate p, which is a user-defined hyperparameter. To generate random binary values with user-defined probability, we design a Bernoulli sampler as illustrated in Figure 5.

The basic hardware module in the proposed Bernoulli sampler is a 4-tap linear feedback shift register (LFSR), which generates a pseudo-random single bit, being 0 or 1, per cycle with a probability 50%. The LFSR is composed of a chain of shift registers formed as a closed loop. The maximum sequence length S_{max} of the LFSR depends on the number of shift registers N_{reg} used in the loop: $S_{max} = 2^{N_{reg}} - 1$. As the N_{reg} is equal to 128 in this work, the used LFSR design would take 1500 years to iterate through the whole sequence when clocked at 160MHz [56]. To perform the Bernoulli sampling with user-defined probability, we deploy N_{lfsr} LFSRs with extra logic, where N_{lfsr} is a reconfigurable hardware parameter specified by users. For instance, to perform the Bernoulli sampling with 25% probability for generating a 1, the design would consist of two LFSRs with an extra AND gate processing the outputs of the two LFSRs each clock cycle. In this paper, we set the maximum value of N_{lfsr} as 5, and thus the minimum Bernoulli probability for keeping an output filter is $p = \frac{1}{2^5}$.

As shown in Figure 4, there can be up to PF dropout masks in each dropout engine and each dropout mask requires one random bit from the Bernoulli sampler. Therefore, we design a serial-in-parallel-out (SIPO) module after the bit-wise logic module to cache and expand the output bitwidth to PF-bits using different random bits.

Since different convolutions in different layers are processed at different speeds, a first-in-first-out (FIFO) buffer is placed at the end of the Bernoulli sampler to cache generated Bernoulli random variables and pop out the mask when required. To improve the hardware performance, we overlap the random mask generation of the Bernoulli sampler with the main execution in the NNE, e.g. convolution or pooling.

4) Smart Buffers: The design of both data and weight buffers needs to consider the support of the previously discussed parallelism in the proposed NNE. As mentioned in Section III-A2, the NNE adopts parallelism in filter (PF), channel (PC) and vector (PV) dimensions. In this work, we propose smart data and weight buffers to support the computation with respect to the three types of parallelism.

Smart Data Buffer — A design overview of the smart data buffer is presented in Figure 6. It mainly consists of a read address generator (RAG), a tree-like fan-out, a crossbar and $PC \times PV$ random access memory (RAM) banks. Before the processing of each layer, the input data is loaded from the off-chip memory, and then cached in the on-chip RAM banks. To generate the read address for each layer, the RAG receives signals such as the kernel size and stride length from the controller, and outputs the read address to RAG. Together with the crossbar and RAG, the RAM banks are able to output $PC \times PV$ data in parallel in a sliding window manner required by both 2D and 3D convolutions. Because one convolutional layer shares the same input feature maps among different filters, we use the tree-like fan-out at the end of the smart data buffer to simply duplicate the outputs by PF times to support the parallelism in the filter dimension, which is implemented through a hand-written hardware module to explicitly improve the timing of the design.

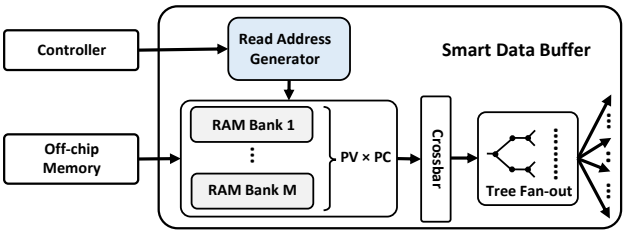


Fig. 6. Hardware design of the smart data buffer.

In terms of the RAM banks, we first separate them into PV groups where each group contains PC RAM banks. Within each bank group, the data arrangement follows the channel-first rule, which means the data from different channels at the same spatial position are stored first, and spanned equally across different banks within one group. Then, the data belonging to the same row are distributed across different bank groups vertically. Following this manner to store one row, we store the rest of data in our smart data buffer horizontally row-by-row and frame-by-frame.

An example of the data arrangement when the input height $H = 2$, the input width $W = 2$, the input channel size $C = 4$ and the temporal length $L = 2$ is illustrated in Figure 7, where the spatial and temporal positions are denoted by different colors and the different channels are represented using $C1{\sim}C4$. As it can be observed, the data with the same color, i.e. position, from different channels are stored first within each bank group. Then, the first and second data in the first row of the first frame are distributed in the first and second bank groups respectively. The rest of data are then stored horizontally row-by-row and frame-by-frame.

Smart Weight Buffer — The design of the smart weight buffer is relatively simple in comparison to the data buffer since weights or the variational parameters do not need to be fetched in the sliding window manner. Therefore, the smart weight buffer only contains $PC \times PF$ FIFOs and a tree-like fan-out as illustrated in Figure 8. The weights of current layer are loaded from the off-chip memory to on-chip FIFOs before the processing for a given layer. Then, FIFOs operated by the controller output $PC \times PF$ weights in parallel to the PE. The fetched weights will flow back to FIFOs for data reuse. Since different data vectors along PV parallelism share the same weights, a tree-like fan-out is designed at the end of the buffer to duplicate the weights PV times. In FIFOs, the weights are stored according to the data arrangement in the smart data buffer described in the previous Section to ensure that the PE receives the corresponding weights.

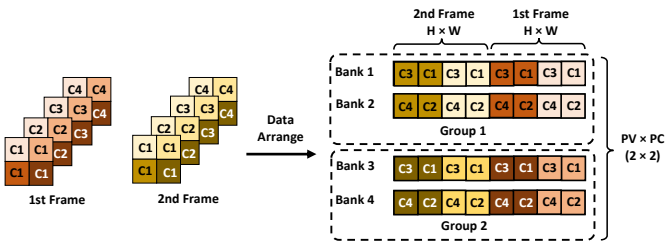


Fig. 7. The data arrangement in the smart data buffer.

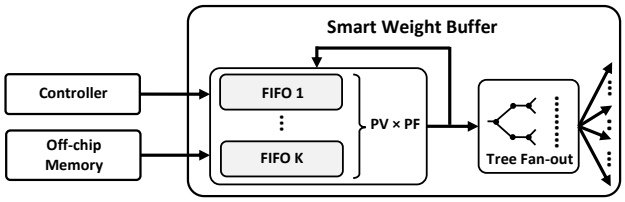


Fig. 8. Hardware design of the smart weight buffer.

B. Mapping 2D and 3D Operations

Based on the hardware accelerator proposed in Section III, we now present a way how to map both 2D and 3D operations into the described unified architecture.

1) 2D and 3D Convolution: As indicated in [57], both 2D and 3D convolutions are the most memory and compute-intensive operations in modern CNNs [2]. To improve the overall performance, we optimize the 2D and 3D convolutions as illustrated in the Algorithm 1. Note that 2D convolution is a special case of 3D Convolution where the temporal length $L, K_L = 1$.

Algorithm 1 Optimized 2D and 3D Convolution.

```
for (l = 0; l < L; l++)
  for (f = 0; f < F/PF; f++)                          ▷ Tiled
    for (h = 0; h < H; h++)
      for (w = 0; w < W/PV; w++)                      ▷ Tiled
        for (k_l = 0; k_l < K_L; k_l++)
          for (k_h = 0; k_h < K_H; k_h++)
            for (k_w = 0; k_w < K_W; k_w++)
              for (c = 0; c < C/PC; c++)              ▷ Tiled
                for (i = 0; i < PC × PV × PF; i++)
                  Perform Convolution
```

Since our hardware design supports three categories of parallelism, i.e. PF, PC and PV, we first tile the filter, channel and width loops so that the innermost loop marked by the blue box in Algorithm 1 can be mapped into the NNE. In the innermost loop, because the results are only accumulated together along the channel dimension, the tiling will produce $PF \times PV$ intermediate results. To avoid caching too many intermediate results, only the computation associated with the $PF \times PV$ intermediate results is performed, such that they can be transferred to off-chip memory as soon as possible. Therefore, we compute the channel and kernel (c, k_h and k_w) loops right after the innermost loop, which is highlighted by the red box in Algorithm 1. The temporal (k_l) loop is set as the outermost loop so that only K_L input frames need to be cached in the on-chip memory. To decrease the memory usage of the smart weight buffer, the convolution is then performed PF filters by PF filters such that only PF filters of weights are cached on-chip each time.

2) 2D and 3D Shortcut Addition: The shortcut (SC) addition was originally proposed in a ResNet pointwise 2D CNN architecture to fast-track gradient propagation and thus achieve better accuracy [2]. Given its success, the SC has been widely adopted in various BayesCNNs to achieve better algorithm performance [58]. While performing SC addition for 2D BayesCNNs, we store a copy of the input data in the SC buffer as illustrated in Figure 9(a). The cached input frame is then added with the output of a convolution to generate the

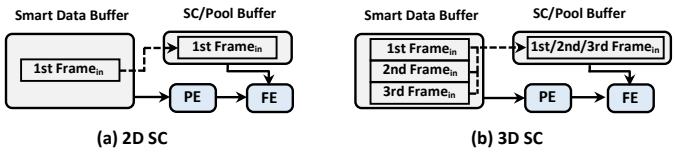


Fig. 9. Hardware implementation of the 2D and 3D SC addition.

final result for that layer. When our design is used to accelerate 3D BayesCNNs, there are K_L consecutive frames cached in the smart data buffer. Instead of caching the entire input, 3D SC addition only caches a single corresponding frame in the SC buffer to reduce the on-chip memory consumption. An example of 3D SC addition when $K_L = 3$ is presented in Figure 9(b), where only one input frame is stored in the SC buffer when required.

3) Other Operations: Similarly to SC addition, the 3D pooling caches the previously-generated results in a buffer. The cached results will then be used together with the outputs of current convolution to perform 3D maximum or average pooling. Note that, the SC buffer is reused as a pooling buffer to reduce the on-chip memory consumption. The batch normalization (BN) [59] is fused into the convolution to reduce computation and memory consumption [60].

In terms of other operations, such as ReLU and MCD, since they do not include any data dependency, 3D operations can be performed by applying their corresponding 2D implementation frame by frame.

IV. Hardware Optimization

A. Overlapping Sampling with Convolution

To improve the overall hardware performance, we propose to overlap the Bernoulli sampling with the computation of the BayesCNN. We observe that the sampling does not depend on the input data, and thus it can be performed independently before or during the evaluation of the BayesCNN. Since generating random binaries for all layers B and samples S will consume too much on-chip memory, we propose an overlapping strategy as shown in Figure 10.

To reduce the on-chip memory consumption, our proposed overlapping strategy only pre-samples Bernoulli random variables for one layer before each 2D or 3D convolutional layer is executed. In this way, since all the random variables required by the currently executing convolutional layer were generated and cached in the on-chip memory, the computation can be executed without any stalls. At the same time, the sampling of the next convolutional layer can be overlapped with the computation of the current layer, which improves the overall hardware performance.

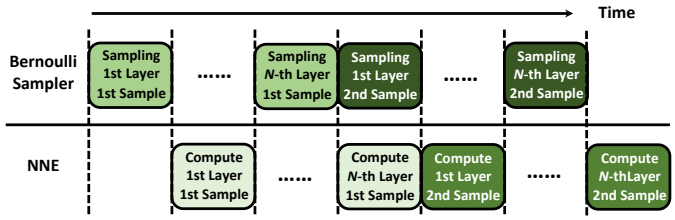


Fig. 10. An example of overlapping sampling with computation. The BayesCNN contains B Bayesian layers and performs $S = 2$ MC samples.

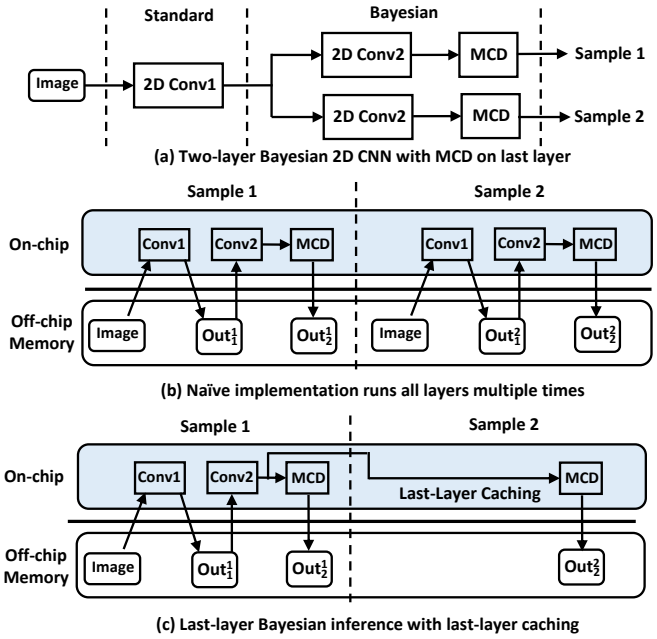


Fig. 11. An example of a two-layer Bayesian CNN which computes two samples to obtain the prediction. The input and output data are denoted by In_j^i and Out_j^i where i means i^{th} iteration and j represents j^{th} layer. The last linear layer is not shown for simplicity.

An example of this procedure is presented in Figure 10. Since our NNE is designed to perform one operation at a time, i.e. convolution, the computation is performed layer-by-layer for each sample sequentially. With the proposed overlapping strategy, all the random variable sampling is performed in advance of the layer that requires it, which avoids any additional latency.

B. Intermediate-layer Caching

As discussed in Section II-B, from an algorithm standpoint, it might not be necessary to place MCD after every layer to achieve fine algorithm performance [10], [36], [38], [61]. Figure 11(a) presents an example of a three-layer 2D BayesCNN with the MCD only applied on the output of the penultimate layer. To perform $S = 2$ MC samples, it is then only required to run the partially Bayesian part B two times instead of inefficiently processing the input through the whole network with depth N. In this work, to avoid the redundant computation and reduce the data transfer between off-chip and on-chip memory, we propose a general hardware-efficient intermediate-layer caching (IC) technique with respect to caching the intermediate results directly in on-chip memory. IC was previously proposed with respect to software caching of the last Bayesian layer [10] and recently extended to the whole network architecture [61]. However, these approaches store the intermediate results in the off-chip memory, which introduces extra data transfer between on-chip and off-chip memory.

In the naive implementation, as shown in Figure 11(b), the evaluation is performed with respect to all the convolutional layers twice and the data is transferred between off-chip and on-chip memory in total eight times. With the IC, as illustrated in Figure 11(c), the output of the second convolutional layer

will be cached on the chip while performing the first MC sample. Then the final result of the second MC sample can be obtained by applying MCD on the cached data, which halves the amount of computation and decreases data transfer up to 5 times. Generally, assuming the NN requires to run the last B layers S times to obtain the prediction, the IC can reduce the compute by $(N - B) \times S$ times and the number of memory accesses by $S \times B$ times.

V. Optimization Framework

A. Workflow of Framework

The reconfigurable accelerator together with partial Bayesian inference provide a large design space for algorithm and hardware performance exploration and trade-off. To explore such a large design space, an automatic framework is proposed to perform the design space exploration for optimizing the algorithm and hardware performance under user-defined constraints and requirements.

The design space in this work contains two categories of configurable parameters, hardware parameters and algorithm parameters. The hardware parameters affect the hardware configuration and they consist of configurable parallelism levels (PF, PC, PV) of the NNE, number of LSFRs in the Bernoulli sampler N_{lfsr} and the memory size of each buffer. The optimization of hardware configurations considers both resource consumption and hardware performance. Since the channel size, filter size, input size or the dropout rate may vary among different BayesCNNs, improper choice of these parameters may lead to unused resources in the hardware, which can decrease the overall hardware performance. At the same time, the recourse consumption of the configured accelerator needs to be smaller than the resource budget the underlying FPGA board provided. Therefore, in our hardware design space, we consider the domains for both PF and PC as $\{8, 16, 32, 64, 128\}$ and PV can be chosen from $\{1, 4, 8, 16\}$. The number of LFSRs N_{lfsr} is an integer ranging from 1 to 5 and the buffers can have arbitrary memory sizes that can be fitted on the given FPGA.

On the algorithm level, there are three parameters that may affect both the algorithmic and hardware performance: the portion of Bayesian layers B supporting partial Bayesian inference mentioned in Section II-B2, the dropout rate p and the number of MC samples S, which represents how many times the B designated layers need to be repetitively run. These three algorithm parameters enable trade-off between latency, accuracy, confidence and uncertainty estimation. For instance, applying MCD only with respect to the penultimate layer and running it with respect to only two MC samples can significantly decrease the latency due to the reduced computation, but may also degrade the quality of uncertainty quantification or the accuracy.

An overview of our proposed framework is presented in Figure 12. There are four metrics optimized by our framework, i.e., algorithm accuracy, hardware latency, uncertainty quantification and calibration confidence. Users are allowed to set maximum or minimum constraints on these four metrics. As mentioned before, since these metrics cover trade-off between

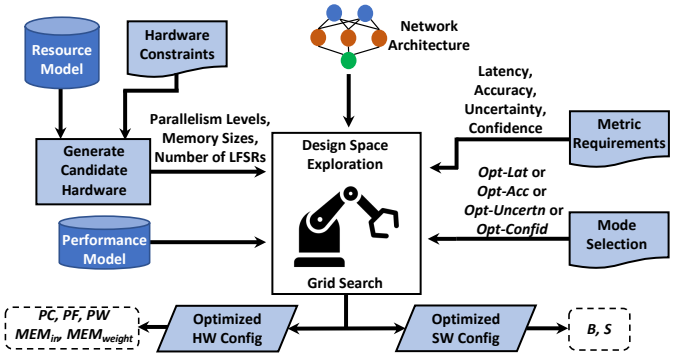


Fig. 12. Overview of the optimization framework.

each other, we design our framework to support four optimization modes, i.e., optimal-latency, optimal-accuracy, optimal-uncertainty quantification and optimal-confidence. These optimization modes indicate the priority to minimise or maximise the chosen objective.

The flow of the framework starts with considering candidate hardware configurations under the hardware constraints using the proposed resource model. Then, the feasible hardware configurations together with a performance model are fed as inputs for the design space exploration. The resource model is used to avoid the time-consuming synthesis and implementation to accelerate the design space exploration. Note that, the estimation of resource consumption is only used during the design space exploration and the final design is implemented and evaluated at the board-level. At the same time, the framework also receives the network architecture of the target BayesCNNs as input. Together with the user-defined constraints and optimization modes, the design space exploration is performed through greedy optimization with respect to software and hardware configurations. The framework optimizes PC, PF, PV, N_{lfsr} and memory sizes as hardware configurations, and B and S as algorithm configurations.

B. Resource Model

The resource model in our paper mainly considers the memory and DSPs resource consumption since they are the limiting resources in FPGA-based NN accelerators [55] and also in our design. The DSPs are mainly consumed by the NNE where one DSP with some extra logic resources are used to implement two 8-bit multipliers for high computational utilization. Hence, the DSP usage in our resource model can be estimated as $DSP = \frac{PC \times PF \times PV}{2}$. The memory consumption mainly accumulates from the FIFOs in the Bernoulli sampler, the weight and input buffers in the NNE. Since there are PF FIFOs in the Bernoulli sampler, its memory consumption can be represented as $MEM_{FIFO} = D_{FIFO} \times PF \times V$, where D_{FIFO} denotes the depth of each FIFO and V represents the bit-width of data. Since the NNE processes the layers in a BayesCNN one-by-one, the memory consumption of the input buffer is dominated by the layer with the maximal input size. Therefore, it can be formulated as $MEM_{in} = \max_{i=1,\ldots,N}(C_i \times H_i \times W_i) \times V$, where H_i, W_i and C_i are the height, width and channel size of the input in the ith layer respectively. As mentioned in Section III, we design the weight buffer to cache

PF filters at a time. Therefore, the memory usage of weight buffer can be modelled as $MEM_{weight} = \max_{i=1,\dots,N}(C_i \times K_H^i \times K_W^i \times K_L^i) \times PF \times V$, where K_H^i, K_W^i, K_L^i are the kernel height, width and temporal length sizes of the ith layer. Since we used the double buffer technique in both weight and input buffers to overlap the loading time between two consecutive layers, the total memory consumption is estimated as $MEM = 2 \times (MEM_{in} + MEM_{weight}) + MEM_{FIFO}$.

VI. Experiments

A. Experimental Setup

We implemented our proposed hardware architecture (Section III) on an Intel Arria 10 SX660 FPGA with a 1GB DDR4 SDRAM installed as off-chip memory. We set PV, PC and PF to be 1, 64 and 64 according to the estimated resource consumption provided by our resource model (Section V-B) and the available resources offered by the underlying hardware. The final design was implemented using Verilog. Quartus 17 Prime Pro was used for synthesis and implementation. Our final design was clocked at 225 MHz. The real resource usage of our accelerator generated by Quartus after the implementation is presented in Table II. We also provide a resource breakdown on two parts: the Bayesian part that is designed to support MCD, and the non-Bayesian part that is used to perform the main computation. We adopted the 8-bit linear quantization [54] for a high hardware performance on our design. [54] demonstrated that 8-bit representation does not have a detrimental effect on BayesCNNs' algorithm performance. To fully utilize the hardware resources, we used one DSP with some extra logic resources to implement two 8-bit multipliers. As mentioned in Section V-B, DSPs are the limiting resource in our accelerator which consume nearly 97% of available DSPs. The non-Bayesian part occupied most of resources compared with the Bayesian part.

Our target applications were image and video classification. In both of these applications, the inputs and the correct classification labels can be represented as tuples $(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ is the input and $\boldsymbol{y}$ denotes the one-hot encoded categories. Given an input $\boldsymbol{x}$ we approximated the predictive distribution according to (1) by averaging the provided output probabilities given by softmax activation at the end of the network with respect to S samples to give $p(\boldsymbol{y}'|\boldsymbol{x})$. Given the predictive distribution, we evaluated the algorithm performance with respect to different metrics. We considered accuracy, average predictive entropy (aPE) for evaluating the quality of quantified uncertainty and expected calibration error (ECE) [62] to measure the calibration of the confidence in the predictions.

TABLE II
Resource utilization of the accelerator on the FPGA.

	Resources	ALMs	Registers	DSPs	M20K
Bayesian Part	Utilization	2%	1%	0%	1%
Non-Bayesian Part	Utilization	69%	51%	97%	85%
Overall Design	Used	303,913	889,869	1,473	2,334
	Total	427,200	1,708,800	1,518	2,713
	Utilization	71%	52%	97%	86%

Note that, our design can be also used to accelerate other applications such as object detection [30] and image segmentation [4] with proper blocking strategies [55].

For the input that should rightfully confuse the net, we measured the quality of the uncertainty prediction with respect to random Gaussian noise with mean and variance of the training data with aPE over a dataset of size E with K classes as: $aPE = \frac{1}{E}\sum_{e=1}^{E} -\sum_{k=1}^{K} p(y_e'^{,k}|\boldsymbol{x}_e)\log p(y_e'^{,k}|\boldsymbol{x}_e)$.

Furthermore, we measured the calibration of the confidence of the BayesCNN with respect to ECE and unmodified test data. ECE quantifies if the BayesCNN is uncalibrated by observing whether the net is making predictions whose confidence are not matching its accuracy. ECE computes a weighted average between accuracy and confidence across bins as: $ECE = \sum_{b=1}^{B} \frac{n_B}{N}|\text{accuracy}(b) - \text{confidence}(b)|$, 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 $B = 10$.

In terms of the datasets, we evaluated the 2D BayesCNNs with respect to image datasets and 3D BayesCNNs with respect to video datasets. To stay consistent with previous designs [16], [18], [19], we considered MNIST [28], SVHN [63] and CIFAR-10/100 [64] for image classification. The datasets were paired with *Bayes-LeNet5* [28], *Bayes-VGG11* [65], and *Bayes-ResNet18/34* [2] respectively. With respect to video datasets we performed the evaluation with respect to UCF-11, video size ranging from $160 \times 120 \times 3$ to $480 \times 360 \times 3$ across several seconds [66],where the first two dimensions stand for the spatial size and the last dimension is the number of color channels, and we used two different architectures: *Bayes-C3D* and *Bayes-R3D18* [15] for evaluation. We picked architectures that are core to different tasks across different applications, e.g. ResNet-like architectures to demonstrate the versatility of this work. We varied both the datasets as well as the architectures to vary the complexity of the experiments from the algorithm and hardware execution perspectives.

In this work, we explored partial Bayesian inference which is motivated by provided hardware execution efficiency. We considered adding dropout at different parts of the networks, always following a 2D or 3D convolutional, BN and ReLU layers, and optionally pooling. We consider $p = 0.25$ for all MCD instances. This value was picked with respect to observing the validation performance across different dataset, architecture and dropout combinations. We considered partial BayesCNNs, such that $B = \{1, \frac{1}{3} \times N, \frac{1}{2} \times N, \frac{2}{3} \times N, N\}$. The number of samples S could be $S = \{3, 4, 5, 6, 7, 8, 9, 10, 20, 50, 100\}$. All experiments were repeated 5 times.

B. Effectiveness of Framework

As introduced in Section V, our proposed framework explores the trade-off between different hardware and algorithm performance metrics, i.e., latency, uncertainty estimation, confidence and accuracy. To meet different users' needs, we design our framework to support four different optimization modes. In this part, we first applied the optimization framework without user constraints to investigate the performance limits of our design under different optimization modes. Then,

TABLE III
THE RESULTANT CONFIGURATIONS, AND THE CORRESPONDING HARDWARE AND ALGORITHM PERFORMANCE OF 2D AND 3D BAYESCNNS UNDER DIFFERENT OPTIMIZATION MODES ON OUR FPGA-BASED ACCELERATOR.

		Opt-Mode	$\{\boldsymbol{B}, \boldsymbol{S}\}$	**Latency** [ms] ↓	**aPE** [nats] ↑	**ECE** [%] ↓	**Accuracy** [%] ↑
Bayesian 2D CNNs	*Bayes-LeNet5*	*Opt-Latency*	1, 3	**0.42**	0.63 ± 0.09	0.25 ± 0.05	99.27 ± 0.04
		Opt-Uncertainty	N, 100	14.83	$\mathbf{1.06 \pm 0.19}$	0.17 ± 0.04	99.32 ± 0.04
		Opt-Confidence	N, 9	1.29	0.98 ± 0.18	$\mathbf{0.1 \pm 0.04}$	99.31 ± 0.03
		Opt-Accuracy	$\frac{2}{3} \times N$, 100	14.32	0.75 ± 0.15	0.13 ± 0.03	$\mathbf{99.39 \pm 0.05}$
	Bayes-VGG11	*Opt-Latency*	1, 3	**0.57**	1.38 ± 0.28	2.8 ± 0.12	95.38 ± 0.1
		Opt-Uncertainty	$\frac{2}{3} \times N$, 100	42.89	$\mathbf{2.02 \pm 0.11}$	0.41 ± 0.05	96.13 ± 0.1
		Opt-Confidence	$\frac{2}{3} \times N$, 100	42.89	2.02 ± 0.11	$\mathbf{0.41 \pm 0.05}$	96.13 ± 0.1
		Opt-Accuracy	N, 100	57.32	1.97 ± 0.05	2.42 ± 0.19	$\mathbf{96.49 \pm 0.05}$
	Bayes-ResNet18	*Opt-Latency*	1, 3	**0.47**	0.36 ± 0.26	4.85 ± 0.19	92.84 ± 0.16
		Opt-Uncertainty	$\frac{1}{2} \times N$, 100	32.04	$\mathbf{1.27 \pm 0.27}$	2.74 ± 0.31	91.12 ± 0.2
		Opt-Confidence	$\frac{2}{3} \times N$, 3	1.20	1.05 ± 0.26	$\mathbf{1.08 \pm 0.06}$	89.99 ± 0.17
		Opt-Accuracy	1, 8	0.50	0.38 ± 0.27	4.74 ± 0.14	$\mathbf{92.91 \pm 0.14}$
	Bayes-ResNet34	*Opt-Latency*	1, 3	**0.93**	0.14 ± 0.15	18.06 ± 0.49	69.63 ± 0.39
		Opt-Uncertainty	N, 100	65.61	$\mathbf{1.97 \pm 0.29}$	14.73 ± 0.33	63.79 ± 0.69
		Opt-Confidence	$\frac{2}{3} \times N$, 3	2.01	1.23 ± 0.65	$\mathbf{1.83 \pm 0.32}$	66.43 ± 0.39
		Opt-Accuracy	$\frac{1}{2} \times N$, 50	19.03	1.1 ± 0.24	3.95 ± 0.44	$\mathbf{71.83 \pm 0.31}$
Bayesian 3D CNNs	*Bayes-C3D*	*Opt-Latency*	1, 3	**93.99**	0.438 ± 0.001	5.23 ± 0.73	85.35 ± 1.1
		Opt-Uncertainty	$\frac{1}{2} \times N$, 100	531.90	$\mathbf{1.443 \pm 0.001}$	3.71 ± 1.65	84.92 ± 1.1
		Opt-Confidence	$\frac{1}{2} \times N$, 100	531.90	1.443 ± 0.001	$\mathbf{3.71 \pm 1.65}$	84.92 ± 1.1
		Opt-Accuracy	1, 50	95.92	0.442 ± 0.001	4.98 ± 0.9	$\mathbf{85.41 \pm 1.2}$
	Bayes-R3D18	*Opt-Latency*	1, 3	**51.01**	0.491 ± 0.001	4.88 ± 0.92	84.63 ± 0.88
		Opt-Uncertainty	$\frac{2}{3} \times N$, 100	1513.42	$\mathbf{1.660 \pm 0.001}$	9.21 ± 0.85	73.88 ± 0.95
		Opt-Confidence	$\frac{1}{3} \times N$, 50	262.62	0.217 ± 0.001	$\mathbf{2.75 \pm 0.57}$	83.43 ± 1.45
		Opt-Accuracy	1, 50	51.22	0.525 ± 0.001	4.57 ± 0.53	$\mathbf{84.89 \pm 1.13}$

we optimized the hardware design as well as the algorithm parameters with respect to user-specific constraints to demonstrate the effectiveness of our framework.

1) Unconstrained Exploration: To determine the global optimal algorithm and hardware performance that our design can achieve, we applied our framework with respect to four different optimization modes: *Opt-Latency*, *Opt-Accuracy*, *Opt-Uncertainty* and *Opt-Confidence*, on all BayesCNNs without any user constraints. We evaluated both 2D and 3D BayesCNNs with different optimized configurations on our FPGA-based accelerators. The results are presented in Table III, the down and up arrows indicate the desired tendency for a given metric.

Under *Opt-Latency* mode, our framework choose the configuration with the minimal number of samples and Bayesian layers, i.e., $\{B = 1, S = 3\}$, for all the 2D and 3D BayesCNNs. In 2D BayesCNNs, our accelerator achieved 0.42ms, 0.57ms, 0.47 and 0.93 ms on *Bayes-LeNet5*, *Bayes-VGG11*, *Bayes-ResNet18* and *Bayes-ResNet34* respectively. Similarly, *Bayes-C3D* and *Bayes-R3D18* needed only 93.99ms and 51.01 ms. With the *Opt-Accuracy* mode enabled, these five BayesCNNs achieved 99.39%, 96.49%, 92.91%, 71.83%, 85.41% and 84.89% accuracy respectively on their corresponding datasets. Under both *Opt-Uncertainty* and *Opt-Confidence* modes, our framework generated different $\{B, S\}$ configurations to achieve better aPE and ECE. For instance, the *Bayes-VGG11* optimized by *Opt-Uncertainty* mode improved the aPE metric by 0.64 compared with the design proposed by *Opt-Latency*. Similarly, *Opt-Confidence* mode reduces the ECE by 2.13% compared with the *Opt-Latency* mode on *Bayes-R3D18*.

2) Constrained Exploration: To validate the effectiveness of our framework in finding the optimal configurations when the user's requirements are provided, we adopted the *Opt-Confidence* mode to optimize *Bayes-ResNet18* on CIFAR-10 dataset with uncertainty, accuracy and latency constraints applied. To demonstrate the optimality of the found configuration, we evaluated all the candidate configurations in terms of latency, aPE, ECE and accuracy. The results are presented in Figure 13. The feasible design space constructed by accuracy, latency and uncertainty constraints is represented by the black box. As we can see, the configuration with the lowest ECE, which is highlighted by the red arrow, is chosen as the optimal configuration under the *Opt-Confidence* mode when user's constraints were given. Therefore, our framework is able to find the optimal configurations with user-specified constraints. In Figure 13, we also visualize the global optimal points generated by *Opt-Latency*, *Opt-Accuracy*, *Opt-Uncertainty* modes without constraints applied, which are highlighted by the black arrows. As we can observe, these global optimal configurations represent the best latency, accuracy and uncertainty that the *Bayes-ResNet18* achieved on its design space.

TABLE IV
PERFORMANCE COMPARISON OF OUR FPGA DESIGN VERSUS CPU AND GPU PLATFORMS.

		CPU		GPU		Our Work			
Platform		Intel Xeon E5-2680 v2		GeForce RTX 2080 Ti		Intel Arria 10 GX1150			
Frequency		2.8 GHz		1.545 GHz		220 MHz			
Technology		22 nm		12 nm		20 nm			
Acceleration Library		MKLDNN, PyTorch 1.9.0		CuDNN, PyTorch 1.9.0		-			
Power [W] ↓		135		248		45			
$\{B, S\}$		$\frac{1}{2} \times N$, 100	$\frac{2}{3} \times N$, 50	$\frac{1}{2} \times N$, 100	$\frac{2}{3} \times N$, 50	$\frac{1}{2} \times N$, 100		$\frac{2}{3} \times N$, 50	
Use IC		Yes [10], [61]		Yes [10], [61]		No	Yes	No	Yes
Latency [ms] ↓	***Bayes-ResNet18***	733.16	583.33	372.92	262.08	44.97	32.04	22.48	18.90
	Bayes-R3D18	5170.0	4920.0	948.27	728.77	5100.7	686.89	2550.3	772.35
Energy Eff.	***Bayes-ResNet18***	0.99	1.57	0.93	1.29	0.020	0.014	0.021	0.017
[J/Sample] ↓	***Bayes-R3D18***	6.98	13.28	2.35	3.61	2.29	0.31	2.30	0.69

TABLE V
PERFORMANCE COMPARISON WITH OTHER STATE-OF-THE-ART ACCELERATORS FOR BAYESIAN NNS.

	ASPLOS'18 [16]	DATE'20 [18]	Micro'20 [19]	Our Work		
Platform	Altera Cyclone V	Zynq XC7Z020	Virtex-7 VC709	Arria 10 GX1150		
Frequency [MHz]	213	200	100	220		
Technology	28 nm	28 nm	28 nm	20 nm		
Available DSPs	342	220	3600	1518		
Power [W] ↓	6.11	2.76	-	43.6		
Model	*Bayes-MLP*	*Bayes-MLP*	*Bayes-GoogLeNet*	*Bayes-VGG11*	*Bayes-ResNet18*	*Bayes-C3D*
Throughput [GOP/s] ↑	59.6	24.22	-	533.75	1590	1449
Speedup Compared with Baseline	-	-	3.1×	49.6×	48.3×	49.9×
Energy Efficiency [GOP/s/W] ↑	9.75	8.77	-	19.6	41.57	34.2
Comp. Eff. [GOP/s/DSP] ↑	0.174	0.121	-	0.362	1.079	0.983

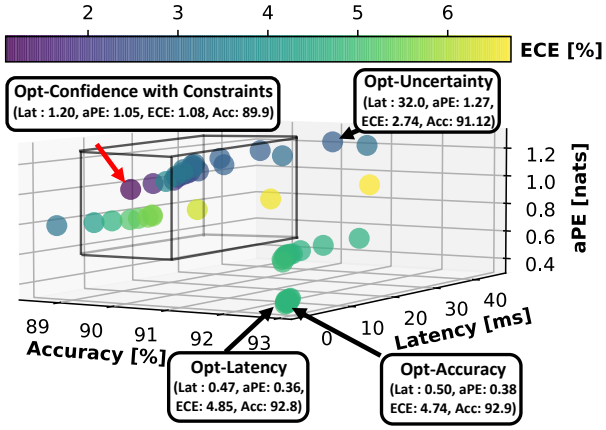


Fig. 13. Design space exploration with latency, accuracy and uncertainty constraints for *Bayes-ResNet18* on CIFAR-10.

C. Performance Comparison against CPU and GPU Implementations

To demonstrate the advantages of our proposed accelerator over other hardware platforms, we measured the hardware performance on the FPGA, Intel Xeon E5-2680 v2 CPU and NVIDIA Titan Xp GPU. Both CPU and GPU versions were implemented using PyTorch 1.9.0 [67]. The CPU implementation was optimized by MKLDNN and the CuDNN library was used to improve the hardware performance of the GPU implementation. As this paper only targets accelerating the evaluation of BayesCNNs, where the inputs are produced sequentially as in real-life scenarios, we set the batch size as 1 for all models on all evaluated hardware platforms. Since PyTorch does not support 8-bit quantization on a GPU, the performance of GPU implementations is reported based on 32-bit floating point. The results are presented in Table IV. We evaluated two configurations for each model, i.e. $\{B = \frac{1}{2} \times N, S = 100\}$ and $\{B = \frac{2}{3} \times N, S = 50\}$. In both CPU and GPU implementations, we enabled the IC optimization by caching the intermediate results of the last non-Bayesian layer as PyTorch tensor variables for the reuse in the following Bayesian layers [10], [61]. To demonstrate the effect of the IC implementation on our accelerator, we evaluated the hardware performance with and without IC optimization on the FPGA.

As we can observe from Table IV, our accelerator achieved different speedup compared with CPU and GPU implementations, depending on the model and configuration. For instance, our design achieved 11∼13 times speedup than the GPU implementations on *Bayes-ResNet18*. At the same time, our design was more energy-efficient than CPU and GPU implementations. For example, we achieved up to 92 and 76 times higher energy efficiency than CPU and GPU implementations on *Bayes-ResNet18* with $\{B = \frac{2}{3} \times N, S = 50\}$. While comparing FPGA implementations with and without on-chip IC, it can be seen that the speedup brought by IC is decreasing when the S becomes smaller and B increases for most of 2D and 3D BayesCNNs. There are three reasons for the achieved higher hardware performance:

- The hardware-level implementation of IC technique that precisely caches the intermediate results of the last non-Bayesian layer in on-chip memory, which decreases the

number of memory accesses and the amount of computation.
- The support for fine-grained parallelism in our design, which fully utilized the extensive concurrency in BayesCNNs.
- The quantization adopted in our design, which reduces the computational complexity and bandwidth requirements.

D. Performance Comparison with Existing Work

To demonstrate the advantages of our hardware architecture and optimization framework over the other state-of-the-art designs, we compared our work with the existing accelerators for Bayesian NNs [16], [18], [19] in Table V. The hardware performance was evaluated in terms of throughput, energy efficiency and compute efficiency for a fair comparison. The energy efficiency was measured in giga-operations per second per watt (GOP/s/W) and the compute efficiency refers to the giga-operations per second per DSP provided (GOP/s/DSP). The results are shown in Table V. As it can be seen, our accelerator was the only design supporting both 2D and 3D BayesCNNs, which demonstrates the versatility of our hardware architecture.

Since both [16] and [18] only support multi-layer perceptrons (MLP), we quoted their performance from the original papers, which was evaluated on a three-layer Bayesian MLP (*Bayes-MLP*). As it can be observed, our design achieved nearly $9\sim30$ and $22\sim66$ times higher throughput than [16] and [18] respectively, depending on the model and configurations. As both [16] and [18] consumed fewer DSPs, we also compared them in terms of the compute efficiency through GOP/s/DSP. It can be seen that our accelerator achieved nearly $2\sim9$ times higher compute efficiency than [16] and [18]. In *Fast-BCNN* proposed by [19], the authors accelerated BayesCNNs by intelligently skipping the zeros generated by MCD and ReLU activation functions. However, they only reported the normalized speedup without mentioning the real hardware performance. Thus, we were not able to compare with them directly. In order to compare with them, we evaluated the speedup brought by our algorithm and hardware optimizations. We first measured the baseline performance on the fully-BayesCNN with IC hardware implementation disabled in our design, which keeps the same configuration as [19]. Then, we measured the speedup by applying the IC and our optimization framework under *Opt-Latency* mode. As observed, our design achieved nearly 16 times higher speedup than [19].

There are three reasons for the higher hardware performance when compared with previous work:
- The partial Bayesian inference together with the hardware-level implementation of IC optimization, which skips the redundant computation in BayesCNNs.
- The fine-grained parallelism and control provided by our hardware architecture.
- The optimization framework with different optimization modes, which optimizes the network configuration for the given BayesCNNs.

VII. Conclusion and Future Work

This work proposes a high-performance FPGA-based design to accelerate 2D and 3D Bayesian convolutional neural networks (BayesCNNs) based on Monte Carlo Dropout. The accelerator is sufficiently versatile to support a variety of 2D and 3D BayesCNNs to achieve up to 4 times higher energy efficiency and 9 times better compute efficiency than other state-of-the-art accelerators. A framework is also proposed to automatically explore the trade-off between algorithm and hardware performance with different priorities, given algorithm requirements and hardware constraints. In future work, we aim to explore neural architecture search and development of a hardware-efficient Bayesian inference scheme that would directly account for hardware performance metrics.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Proceedings of the 2017 Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[2] 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)*, 2016, pp. 770–778.

[3] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[4] M. Ferianc, D. Manocha, H. Fan, and M. Rodrigues, "Combinet: Compact convolutional Bayesian neural network for image segmentation," *arXiv preprint arXiv:2104.06957*, 2021.

[5] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object detection via region-based fully convolutional networks," in *Proceedings of the 2016 Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 379–387.

[6] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (CVPR)*, 2015, pp. 4489–4497.

[7] K. Hara, H. Kataoka, and Y. Satoh, "Can spatiotemporal 3D CNNs retrace the history of 2D CNNs and Imagenet?" in *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 6546–6555.

[8] H. Lu, H. Wang, Q. Zhang, S. W. Yoon, and D. Won, "A 3D convolutional neural network for volumetric image semantic segmentation," *Procedia Manufacturing*, vol. 39, pp. 422–428, 2019.

[9] 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.

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

[11] R. M. Neal, "Bayesian learning via stochastic dynamics," in *Proceedings of the 1993 Advances in Neural Information Processing Systems (NeurIPS)*, 1993, pp. 475–482.

[12] 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)*, 2016, pp. 1050–1059.

[13] M. de la Riva and P. Mettes, "Bayesian 3D convnets for action recognition from few examples," in *Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, 2019, pp. 0–0.

[14] 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.

[15] H. Fan, X. Niu, Q. Liu, and W. Luk, "F-C3D: FPGA-based 3-Dimensional convolutional neural network," in *Proceedings of the 2017 International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–4.

[16] 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.

[17] T. Myojin, S. Hashimoto, and N. Ishihama, "Detecting uncertain BNN outputs on FPGA using Monte Carlo Dropout sampling," in *Proceedings of the 2020 International Conference on Artificial Neural Networks (ICANN)*. Springer, 2020, pp. 27–38.

[18] 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)*, 2020, pp. 1402–1407.

[19] 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.

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

[21] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170.

[22] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, pp. 1354–1367, 2018.

[23] Y. Xing, S. Liang, L. Sui, X. Jia, J. Qiu, X. Liu, Y. Wang, Y. Shan, and Y. Wang, "Dnnvm: End-to-end compiler leveraging heterogeneous optimizations on FPGA-based CNN accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[24] E. Wang, J. J. Davis, R. Zhao, H.-C. Ng, X. Niu, W. Luk, P. Y. Cheung, and G. A. Constantinides, "Deep neural network approximation for custom hardware: Where we've been, where we're going," *arXiv preprint arXiv:1901.06955*, 2019.

[25] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on FPGAs," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[26] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 26–35.

[27] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to FPGAs," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[29] M. Ferianc, H. Fan, and M. Rodrigues, "Vinnas: Variational inference-based neural network architecture search," *arXiv preprint arXiv:2007.06103*, 2020.

[30] H. Fan, S. Liu, M. Ferianc, H.-C. Ng, Z. Que, S. Liu, X. Niu, and W. Luk, "A real-time object detection accelerator with compressed SSDLite on FPGA," in *Proceedings of the 2018 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2018, pp. 14–21.

[31] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2012.

[32] R. Hou, C. Chen, R. Sukthankar, and M. Shah, "An efficient 3D CNN for action/object segmentation in video," *arXiv preprint arXiv:1907.08895*, 2019.

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

[34] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: representing model uncertainty in deep learning," *arXiv preprint arxiv:1506.02142*, 2015.

[35] 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.

[36] 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.

[37] E. Daxberger, E. Nalisnick, J. U. Allingham, J. Antorán, and J. M. Hernández-Lobato, "Expressive yet tractable Bayesian deep learning via subnetwork inference," *arXiv preprint arXiv:2010.14689*, 2020.

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

[39] G. Ofenbeck, R. Steinmann, V. Caparros, D. G. Spampinato, and M. Püschel, "Applying the roofline model," in *Proceedings of 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2014, pp. 76–85.

[40] H. Fan, C. Luo, C. Zeng, M. Ferianc, Z. Que, S. Liu, X. Niu, and W. Luk, "F-E3D: FPGA-based acceleration of an efficient 3D convolutional neural network for human action recognition," in *Proceedings of the IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160. IEEE, 2019, pp. 1–8.

[41] K. Hegde, R. Agrawal, Y. Yao, and C. W. Fletcher, "Morph: Flexible acceleration for 3D CNN-based video understanding," in *Proceedings of the 2018 Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 933–946.

[42] J. Shen, Y. Huang, Z. Wang, Y. Qiao, M. Wen, and C. Zhang, "Towards a uniform template-based architecture for accelerating 2D and 3D CNNs on FPGA," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2018, pp. 97–106.

[43] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4013–4021.

[44] Z. Liu, P. Chow, J. Xu, J. Jiang, Y. Dou, and J. Zhou, "A uniform architecture design for accelerating 2D and 3D CNNs on FPGAs," *Electronics*, vol. 8, no. 1, p. 65, 2019.

[45] T. Liang, J. Glossner, L. Wang, and S. Shi, "Pruning and quantization for deep neural network acceleration: A survey," *arXiv preprint arXiv:2101.09671*, 2021.

[46] 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 (ICPR)*, 2018, pp. 2704–2713.

[47] H. Fan, H.-C. Ng, S. Liu, Z. Que, X. Niu, and W. Luk, "Reconfigurable acceleration of 3D-CNNs for human action recognition with block floating-point representation," in *Proceedings of the 2018 International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 287–2877.

[48] H. Fan, G. Wang, M. Ferianc, X. Niu, and W. Luk, "Static block floating-point quantization for convolutional neural networks on FPGA," in *Proceedings of the 2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 28–35.

[49] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017, pp. 75–84.

[50] M. Sun, P. Zhao, M. Gungor, M. Pedram, M. Leeser, and X. Lin, "3D CNN acceleration on FPGA using hardware-aware pruning," in *Proceedings of the 2020 ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[51] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, and D. Chen, "FPGA/DNN co-design: An efficient design methodology for 1ot intelligence on the edge," in *Proceedings of the 2019 ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[52] H. Fan, M. Ferianc, S. Liu, Z. Que, X. Niu, and W. Luk, "Optimizing FPGA-based CNN accelerator using differentiable neural architecture search," in *Proceedings of the 2020 IEEE International Conference on Computer Design (ICCD)*. IEEE, 2020, pp. 465–468.

[53] X. Jia, J. Yang, R. Liu, X. Wang, S. D. Cotofana, and W. Zhao, "Efficient computation reduction in Bayesian neural networks through feature decomposition and memorization," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[54] M. Ferianc, P. Maji, M. Mattina, and M. Rodrigues, "On the effects of quantisation on model uncertainty in bayesian neural networks," *arXiv preprint arXiv:2102.11062*, 2021.

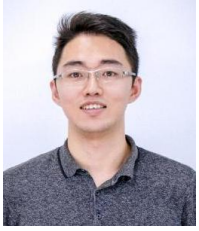
[55] S. Liu, H. Fan, X. Niu, H.-c. Ng, Y. Chu, and W. Luk, "Optimizing CNN-based segmentation with deeply customized convolutional and deconvolutional architectures on FPGA," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–22, 2018.

[56] R. Andraka and R. Phelps, "An FPGA based processor yields a real time high fidelity radar environment simulator," in *Proceedings of the Military and Aerospace Applications of Programmable Devices and Technologies Conference*, 1998, pp. 220–224.

[57] G. Habib and S. Qureshi, "Optimization and acceleration of convolutional neural networks: A survey," *Journal of King Saud University-Computer and Information Sciences*, 2020.

[58] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5455–5516, 2020.

[59] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.
[60] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.
[61] 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.
[62] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," *arXiv preprint arXiv:1706.04599*, 2017.
[63] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proceedings of the 2011 NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. [Online]. Available: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf
[64] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.
[65] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[66] K. Soomro and A. R. Zamir, "Action recognition in realistic sports videos," in *Computer vision in sports*. Springer, 2014, pp. 181–208.
[67] 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.

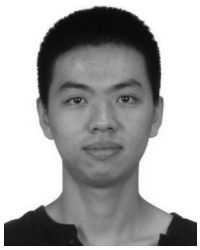


Hongxiang Fan received the B.S. degree in electronic engineering from Tianjin University, Tianjin, China, in 2017, and the master's degree from the Department of Computing, Imperial College London, London, U.K., in 2018. He is currently a Ph.D. student in Machine Learning and High-Performance Computing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

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 tools for hardware design optimization.

Shuanglong Liu received the B.Sc. and M.Sc. degrees from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2010 and 2013 respectively, and Ph.D. degree in Electric Engineering from Imperial College London, London, U.K, in 2017. From 2017 to 2020, he was a Research Associate with the Department of Computing, Imperial College London. He is currently a Distinguished Professor in the School of Physics and Electronics, Hunan Normal University, Changsha, China.



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. He is a member of the IEEE Signal Processing Society Technical Committee on "Signal Processing Theory and Methods", and the EURASIP SAT on Signal and Data Analytics for Machine Learning.

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.