Reconfigurable Acceleration of 3D-CNNs for Human Action Recognition with Block Floating-Point Representation

Hongxiang Fan*, Ho-Cheung Ng, Shuanglong Liu§, Zhiqiang Que, Xinyu Niu, Wayne Luk
Dept. of Computing, School of Engineering, Imperial College London, UK
Email: {h.fan17, h.ng16, s.liu13, z.que, niu.xinyu10, w.luk}@imperial.ac.uk

***Abstract*—Human action recognition (HAR) has been widely employed in various applications such as autonomous cars and intelligent video surveillance. Among the algorithms proposed for HAR, the 3D-CNNs algorithm can achieve the best accuracy. However, its algorithmic complexity imposes a substantial overhead over the speed of these networks, which limits their deployment in real-life applications. This paper proposes a novel customizable architecture for 3D-CNNs based on block floating-point (BFP) arithmetic, where the utilization of BFP significantly reduces the bitwidth while eliminating the need to retrain the network. Optimizations such as locality exploration and block alignment with 3D blocking are performed to improve performance and accuracy. An analytical model and tool are developed to predict the optimized parameters for hardware customization based on user constraints such as FPGA resources or accuracy requirement. Experiments show that without retraining, a 15-bit mantissa design using single-precision accumulation on a Xilinx ZC706 device can be 8.2 times faster than an Intel i7-950 processor at 3.07 GHz with only 0.4% accuracy loss.**

I. Introduction

Recent technological advancements and cost reduction of cameras have resulted in a huge demand for human action recognition (HAR) in various domains such as surveillance, assisted living, autonomous vehicle, etc. Very often, these cameras are installed in an environment with limited bandwidth and power budget, and it is therefore imperative to have HAR handled in the vicinity of the cameras with embedded devices to reduce data transfer and power consumption [1].

Deep 3-dimensional convolutional networks (3D-CNNs) have demonstrated their outstanding classification performance in HAR compared to other algorithms. However, the algorithmic complexity and memory bandwidth demands of 3D-CNNs impose a huge overhead on the processing speed. Therefore, different hardware devices such as FPGAs, ASIC and GPUs have been utilized to accelerate 3D-CNNs for HAR. In particular, FPGAs are gaining popularity because of their better reconfigurability and shorter turn-around time than ASICs and higher energy efficiency than GPUs.

In spite of these advantages, there are several challenges when accelerating 3D-CNNs on FPGA for HAR:

1) Compared to 2-dimensional convolutional networks (2D-CNNs), 3D-CNNs require a lot more memory and computational resources since they consist of a large number of layers with a massive amount of computations. Widely studied approach to FPGA implementation of 2D-CNNs may not be suitable for 3D-CNNs [2].
2) FPGA implementations of 2D-CNNs tend to utilize low bitwidth fixed-point arithmetic to increase the overall throughput, and retraining is required to guarantee the classification accuracy. However, proper selection of bitwidth and the subsequent retraining process usually necessitate days of analysis and computation for 3D-CNNs.
3) The accuracy loss after quantization varies in different neural networks [3]. In particular, previous work on FPGA acceleration of 3D-CNNs with fixed-point arithmetic does not showcase reasonable classification accuracy [4].

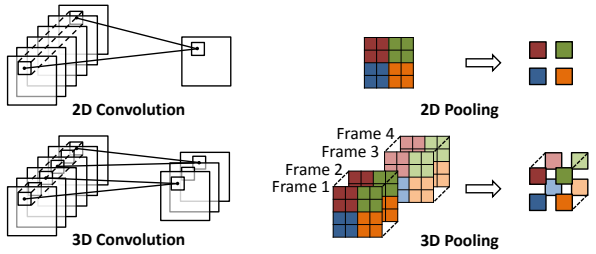
To address the above challenges, we introduce a novel FPGA architecture of 3D-CNNs based on block floating-point (BFP) arithmetic [5], where the use of BFP can reduce the bitwidth while eliminating the need of retraining. A thorough exploration of the mantissa bitwidth and accumulation methods for BFP with respect to the accuracy and performance is also presented. BFP adopts fixed-point arithmetic to emulate floating-point (FP) representation by assigning a block of data with an exponent instead of having an individual exponent for each data. The advantages of using BFP in 3D-CNNs are twofold: First, it significantly reduces the memory and DSP requirements compared to the standard FP calculation. Second, as all the calculations are still based on FP arithmetic, the network can offer competitive classification accuracy without the need to retrain the network, which can take more than 4 to 5 days even on a TITAN X GPU. This is important because nowadays a new network model can appear within months.

Moreover, we provide an automatic tool that determines the optimal parameters to customize the proposed 3D-CNNs architecture based on the given constraints including the resource budget and the accuracy requirement. This also ensures the scalability of the 3D-CNNs accelerator when the hardware design is implemented onto a larger FPGA.

The main contributions of this work are the following:

- A customizable FPGA implementation of 3D-CNNs that performs fast and accurate HAR with BFP which eliminates the need to retrain the network (Section III).
- Different optimization strategies including locality optimization and block alignment with 3D blocking which improve

* The first author is financially supported by China Scholarship Council.
§ Corresponding author.



(a) (b)

Fig. 1: A comparison of the 2D and 3D convolution and pooling layers with pooling size $2 \times 2 \times 2$.

TABLE I: Parameters used in the FPGA implementation of 3D-CNNs for HAR acceleration.

Parameter	Description
H	the height of input feature map
W	the width of input feature map
K_c	the kernel size of 3D convolution
K_p	the kernel size of 3D pooling
S	the stride of 3D convolution kernel
N_c	the number of channels
N_f	the number of filters
N_l	the number of frames

both the performance and accuracy (Section III).

- An experimental analysis of single-precision (SP) versus double-precision (DP) accumulation of BFP demonstrating that SP can offer at least 2 times speedup with identical accuracy compared to DP (Section V).
- An automatic tool that determines the optimal parameters for network customization to balance the performance-accuracy trade-off based on user constraints (Section IV).

II. Background

This section provides an overview of the 3D-CNNs employed in the proposed implementation for HAR acceleration by making an analogy to traditional 2D-CNNs. Also, a comprehensive description of BFP arithmetic and its implications on FPGA are given in addition to a review of related work.

A. 3-dimensional Convolutional Networks

1) 3-dimensional Convolution and Pooling: Basically, 3D-CNNs have the capability of incorporating the third dimension information into the analysis. Compared to 2D-CNNs, this results in a difference in the convolution and the pooling layer.

Figure 1a displays a comparison of the 2D and 3D convolution layers. Assume that the input data consists of multiple frames, 2D convolution simply accumulates all the results from different video frames and outputs one image when it is used in HAR. However, 3D convolution preserves the temporal information in all different video frames and generates an output volume, which is a collection of frames. To provide a better illustration, a detailed explanation of 3D convolution layer is presented in Algorithm 1 where the notations and parameters are given by Table I. Note that the same set of parameters are also adapted in the rest of this paper.

Similar to 3D convolution, 3D pooling will also perform calculations along the adjacent video frames. Figure 1b presents an example of 3D pooling with pooling size $2 \times 2 \times 2$ in which the pixels with the same color produces one output pixel. This decreases the number of frames and the size of the feature maps as the network goes deeper, which is similar to the pooling layers in 2D-CNNs.

Algorithm 1 Pseudocode of 3D Convolution.

```
1: for channels = 1 to N_c do
2:     for filters = 1 to N_f do
3:         for frames = 1 to (N_l + 2 × P − K_c + 1)/S do
4:             for i = frames to frames + K_c do
5:                 output_fm[filters][frames]+=
6:                 coef[filters][channels]×input_fm[channels][i]
```

B. Block Floating-Point (BFP) Arithmetic

1) Representation: Similar to floating-point (FP), BFP representation utilizes a mantissa and an exponent to represent a wide range of value. However, BFP separates the data into different blocks. The numbers in the same block have a joint scaling factor that corresponds to the largest exponent value within that block. Hence, the memory required to store the data of BFP can be significantly reduced. Figure 2 illustrates the representation of FP and BFP and their memory usage.

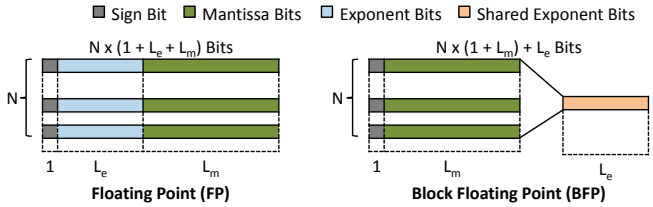


Fig. 2: Representation and the memory usage of FP and BFP.

The block size and the number distribution are the two major factors that affect the precision loss of BFP. As numbers belonging to one block share the same exponent value, the mantissa of each number needs to be shifted to align. When the variance of numbers in one block is very large, the shifting can be greater than L_m which causes severe precision loss. Furthermore, the variance between the numbers within a block may become larger when the block size increases. As a result, a smaller block size can contribute to much less precision loss.

2) Inner Product: The inner product of BFP can be separated into two parts: summation and multiplication of the mantissas and addition of the exponents. Assume that $\mathbf{A}$ and $\mathbf{B}$ are two vectors, the BFP representations of $\mathbf{A}$ and $\mathbf{B}$ are

$$\mathbf{A} = \mathbf{M}_{\mathbf{A}}^{'} \times 2^{\xi_{\mathbf{A}}}, \qquad \mathbf{B} = \mathbf{M}_{\mathbf{B}}^{'} \times 2^{\xi_{\mathbf{B}}},$$

where $\mathbf{M}_{\mathbf{A}}^{'}$ and $\mathbf{M}_{\mathbf{B}}^{'}$ are the aligned mantissas of $\mathbf{A}$ and $\mathbf{B}$, $\xi_{\mathbf{A}}$ and $\xi_{\mathbf{B}}$ are the largest exponents in $\mathbf{A}$ and $\mathbf{B}$ respectively. The result of the inner product $c = \mathbf{M}_{\mathbf{A}}^{'}\mathbf{M}_{\mathbf{B}}^{'}$ is given by:

$$c = [m_{1,a}^{'} \times m_{1,b}^{'} + m_{2,a}^{'} \times m_{2,b}^{'} + ... + m_{N,a}^{'} \times m_{N,b}^{'}] \times 2^{\xi_{\mathbf{A}}+\xi_{\mathbf{B}}},$$

where $m_{i,a}^{'}$ and $m_{i,b}^{'}$ are the ith elements in $\mathbf{M}_{\mathbf{A}}^{'}$ and $\mathbf{M}_{\mathbf{A}}^{'}$ respectively. There are two methods to perform the summation and multiplication of the mantissas: double-precision (DP)

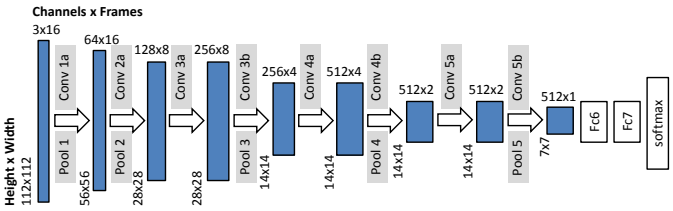


Fig. 3: C3D architecture with eight convolution layers, five max pooling layers and two fully connected layers.

or single-precision (SP) accumulation. In DP, the bitwidths of multipliers, adders and any intermediate results are all $1+2\times L_m$. Truncation only occurs in the final accumulated result. In SP, $1+L_m$ bitwidth adders and multipliers are utilized and truncation is performed after every addition and multiplication. Although the implementation of DP can achieve much less precision loss, it requires more computational resources. Therefore, an evaluation of DP versus SP is done to determine the trade-off between resource consumption and accuracy.

C. Related Work

Previous FPGA implementations for HAR are mainly based on simple algorithms such as Histogram of Oriented Gradient in 3D [1] or Support Vector Machine [6]. However, the performance and accuracy can be disappointing if an enormous class of actions is required to classify, especially when only a small dataset with limited classes of human action is used in their evaluations.

C3D [7], which is one of the most commonly used 3D-CNNs for HAR, is employed as the network model in the proposed FPGA implementation. This network can achieve 85.2% recognition accuracy on Sports-1M dataset [8], which consists of 487 classes of actions. The details about the C3D network is shown in Figure 3. The network takes videos as inputs by splitting them into 16 non-overlapped frame clips and resizing them into 112×112 with 3 channels. The network produces the predicted actions as the classification outputs.

Although a substantial amount of efforts has been devoted to accelerating CNNs on FPGA [9], only a few of them study the reconfigurable acceleration of C3D. In [4], the authors propose an FPGA implementation of C3D based on 16-bit fixed-point representation. Pixel blocking is used to maximize the data reuse and minimize the memory overhead. However,the data locality has not been fully explored and the accuracy is only 55.1% on Sports-1M dataset [8] which is unacceptable for many real-life applications. Shen et al. [2] also develop an FPGA design for accelerating C3D and the architecture is based on a uniform template. 3D Winograd algorithm is used to reduce the arithmetic complexity of convolution and 16-bit fixed-point is adapted for evaluations. Despite the high performance with 430.7 Gops on VC709, the accuracy of the accelerator is unexplored and its usefulness in HAR is unclear.

While BFP has been used in FPGA designs for many years [10], its error analysis for CNNs has only been presented recently. In [11], Köster et al. utilize 16-bit BFP (Flexpoint) to train 2D-CNNs with only negligible accuracy loss. Song et al. [12] present their error analysis for 2D-CNNs based on Caffe framework [13]. However, these works are only implemented on the software level for 2D-CNNs. Although Aydonat et al. [14] utilize shared exponent technique in their Deep Learning Accelerator (DLA) to accelerate 2D-CNNs, there is no thorough exploration for the mantissa bitwidth and BFP accumulation methods in regard to the accuracy and performance from the hardware perspective.

This paper extends the accuracy evaluation of BFP to 3D-CNNs, provides a detailed investigation for BFP accumulation method, and proposes the first BFP-based 3D-CNNs on FPGA. Compared to the previous work, our proposed, customizable FPGA design of C3D can achieve 84.8% accuracy with 33 frames per second (fps) which are sufficient for HAR in many real-life applications.

TABLE II: Number of transfers required and memory usage for the input and output data based on different loop sequences.

Loop Sequence *(Outermost→Innermost)*	No. of Trans.	Memory Usage of Input Data	Memory Usage of Output Data
FILTER-CHANNEL-FRAME	1	$N_c \times N_l \times H \times W \times bw$	$N_l \times H \times W \times bw$
FILTER-FRAME-CHANNEL	N_l	$N_c \times N_l \times H \times W \times bw$	$K_c \times H \times W \times bw$
FRAME-FILTER-CHANNEL	N_l^2	$N_c \times H \times W \times bw$	$K_c \times N_f \times H \times W \times bw$
FRAME-CHANNEL-FILTER	N_l^2	$N_c \times H \times W \times bw$	$K_c \times N_f \times H \times W \times bw$
CHANNEL-FILTER-FRAME	1	$N_l \times H \times W \times bw$	$N_f \times N_l \times H \times W \times bw$
CHANNEL-FRAME-FILTER	N_l	$N_l \times H \times W \times bw$	$N_f \times N_l \times H \times W \times bw$

III. FPGA Accelerator Design

In this section, locality optimization of the network coefficients is first presented by comparing different loop sequences and data access patterns. Then, the implementation of the 3D-CNNs accelerator which targets at SoC platform is described by introducing each of the modules within the design and their interactions in detail.

A. Locality Optimization for Network Coefficients

In the 3D convolution layer, the number of network coefficients is $N_c \times N_l \times (K_c)^3$ which is too large to be stored in the on-chip memory on FPGA. Therefore, only portions of the coefficients data can be cached in the block RAMs while the rest has to be stored in the off-chip memory. As the bandwidth between the on-chip and off-chip memory is limited, it is desirable to explore the optimal data access pattern to improve the locality of the network coefficients and to minimize the number of data transfer.

As shown in Algorithm 1, there are 4 loops in total. Different loop sequences result in different data access patterns which in turn affects the locality of the network coefficients and on-chip memory usage. To simplify the following analysis, we first combine the innermost loop and the loop with the variable $frames$ together as the FRAME loop, as K_c is a small constant and the variable i is correlated to the variable $frames$. The loops with the variables $channels$ and $filters$ are denoted as the CHANNEL and FILTER loop respectively.

Table II summarizes the number of transfers required for the entire network coefficients based on different loop sequences. The corresponding memory usage for the input and output

data are also displayed in the same table. The leftmost column indicates the loop sequence from the outermost to innermost loop, and symbol bw represents the bitwidth of a pixel.

Essentially, the number of transfers required for the entire network coefficients (size: $N_c \times N_l \times (K_c)^3$) is minimum when FILTER-CHANNEL-FRAME or CHANNEL-FILTER-FRAME are adopted. The memory usage for FILTER-CHANNEL-FRAME requires $(N_c \times N_l + N_l) \times H \times W \times bw$ bits while CHANNEL-FILTER-FRAME consumes $(N_f \times N_l + N_l) \times H \times W \times bw$ bits of memory. As N_c is smaller than or equal to N_f, FILTER-CHANNEL-FRAME is chosen as the access sequence for the accelerator. Algorithm 2 illustrates the pseudo-code of the chosen sequence FILTER-CHANNEL-FRAME, which is denoted as frame-major access pattern in the rest of this paper for simplicity.

Algorithm 2 3D Convolution with FILTER-CHANNEL-FRAME

```
1: for filters = 1 to N_f do
2:     for channels = 1 to N_c do
3:         for frames = 1 to (N_l + 2 × P − K_c + 1)/S do
4:             for i = frames to frames + K_c do
5:                 output_fm[filters][frames]+=
6:                 coef[filters][channels]×input_fm[channels][i]
```

B. Blocking and Block Alignment

As the accelerator is based on an SoC platform, we propose a specific access and grouping mechanism onto the input volumes so as to simplify the datapath implementation and to increase the amount of parallelism. Basically, every frame in the input volume is divided into equal blocks (*blocking*). Blocking occurs at the same position with the same size on every frame along the input volume. In other words, each set of blocks which shares the same positions forms an individual block volume. Then, each number in the block volume is aligned based on the largest exponent in its belonging block (*block alignment*), forming a BFP representation. Note that the entire blocking and block alignment process is handled on the processor, right before a block volume being transferred to the on-chip memory for temporary buffering. Figure 4 provides an illustration of the entire process.

The decision to employ a specific access and grouping mechanism is mainly a consideration to improve its accuracy and performance. To start, when a calculation is performed over a small subset of the original input volume each time, every block volume can be buffered in the on-chip memory for low-latency access. Furthermore, the hardware design, such as the exponent module, can be simplified when block alignment is handled within each individual block. For example, the exponent in 3D convolution can be added directly since the same block of data is now aligned. A simpler design can contribute to a lower area cost of each module which in turn increases the number of parallelisms. Finally, the alignment of BFP numbers is based on the individual block instead of the entire block volume or frame because it can minimize the precision loss.

Since different 3D convolution layers have different settings in terms of block size, the output of a convolution layer needs to be realigned to fit the setting of the next layer. The realignment procedure, which is performed on the processor, can completely overlap with the computations on FPGA as presented in Figure 5.

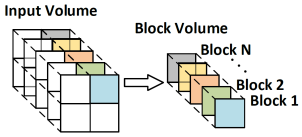


Fig. 4: 3D blocking occurs at the same position with the same size on every frame along the input volume.

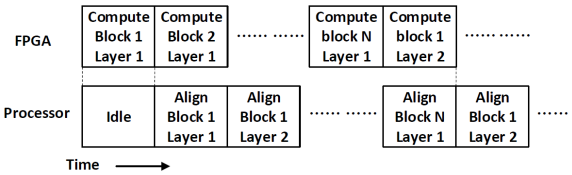


Fig. 5: The realignment procedure can completely overlap with the computations on the FPGA.

The major challenge of blocking and its alignment is the proper selection of the block size for each layer. On the one hand, a smaller block size can improve the accuracy, but it also decreases the performance. For example, the arithmetic becomes floating-point when the block size is 1. On the other hand, although a larger block size can reduce the memory usage, it also increases accuracy loss. Therefore, we will elaborate the optimization of block size in Section IV.

C. System Overview

Figure 6 presents a system overview of the FPGA accelerator. It consists of multiple modules that implement the 3D convolution layer (`3D Conv`) with pooling layer (`3D Pool`) and a fully-connected layer (`FC`) based on [4]. In particular, various layers with different parameters are achieved with one layer of `3D Conv` and `FC` on-chip. The parameters of different layers are specified using the processor onboard. The processor is also responsible for miscellaneous control and alignment of the BFP numbers. Direct Memory Access is used to transfer the network coefficients, input frames and output actions to/from FPGA.

D. 3D Convolution

Figure 7 presents the design of the 3D convolution and 3D pooling. The same figure also indicates their interactions between other hardware modules/sub-modules. The size of the convolution kernel is set to be $3 \times 3 \times 3$ as it is revealed to be optimal according to [7].

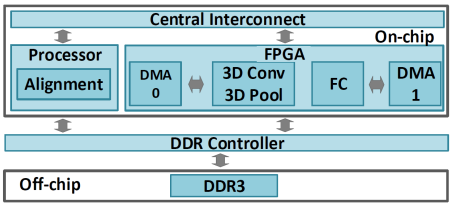


Fig. 6: System overview of the FPGA accelerator.

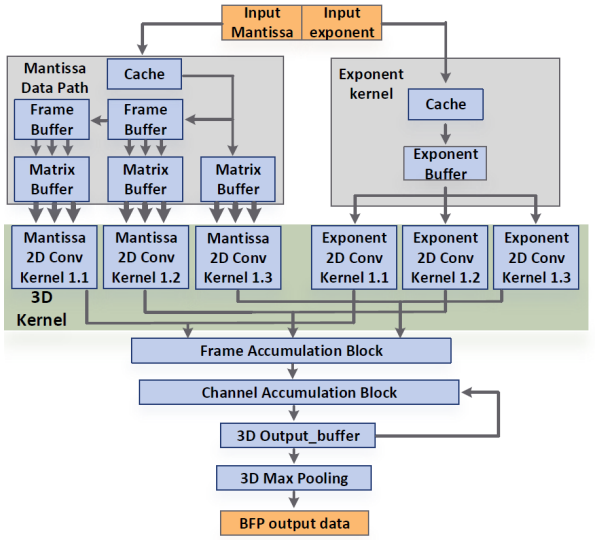


Fig. 7: Hardware architecture of 3D convolution and its interaction with other modules/sub-modules.

As mentioned, only a block volume is transferred from the off-chip memory to the FPGA in each iteration of computation. A block within the transferred block volume is considered to be a frame which consists of pixels in BFP format. Each pixel is split into mantissa and exponent and they are directed to their corresponding datapaths for temporary buffering. As the dot product of BFP can actually be separated into multiplication and addition of the mantissa and addition of exponent, the proposed architecture contains separated datapaths and convolution kernels for the calculation of the mantissa and exponent respectively.

Different buffers are developed to control the dataflow of the mantissa and exponent data. In the datapath of the mantissa, two frame buffers are developed to cache 2 contiguous input frames. When the $(i+2)$th frame flows from the line buffer to matrix buffer, the ith and $(i+1)$th frames cached in the frame buffers flows also out. These buffers are needed because the kernel size is now set to $3 \times 3 \times 3$, and 3D convolution needs to accumulate 3 contiguous frames. The matrix buffers accept the pixels from the frame buffers and output the 3×3 pixels needed by the sliding window of the convolution kernel. With this mechanism, three 2D mantissa kernels can receive the data simultaneously, which simplifies the control logic. The 2D mantissa kernel is responsible for the convolution operation onto the frames which essentially computes a dot product between the kernel weights and the 3×3 pixels from the matrix buffer.

In the datapath of the exponent, since calculation of the exponent only requires addition, only one exponent buffer which contains three FIFO is needed to store the exponents of three contiguous input frames.

3D Kernels — Since the convolution kernel is $3 \times 3 \times 3$ in size, the 3D convolution kernel consists of three 2D exponent kernels and three 2D mantissa kernels. Also because of the blocking and block alignment procedure in Section III-B, the

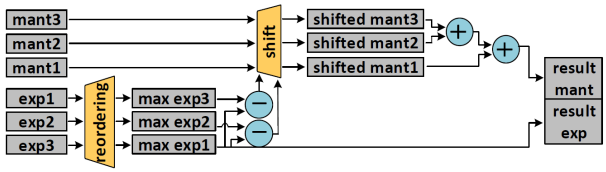


Fig. 8: The hardware architecture of the accumulator.

2D exponent kernel only consists of a L_e-bit fixed-point adder which is used to perform exponent addition. On the other hand, the 2D mantissa kernel is composed of 3×3 multipliers and a fully-pipelined tree adder with $\lceil \log_2 (3 \times 3) \rceil$ levels, where $\lceil \rceil$ is the ceiling function. As mentioned, the inner product can be implemented with single precision (SP) or double precision (DP) accumulation, and this affects the bitwidth used by multipliers and adders in the 2D mantissa kernel. In the next section, we will explore the performance and accuracy of these two modes of calculations.

Accumulator — The major component of the submodule Frame Accumulation and Channel Accumulation is the accumulator displayed in Figure 8. These two submodules correspond to the operations in Line 5 of the frame-major access as shown in Algorithm 2. Essentially, reordering is first performed on the number to find the maximal exponent. Then the accumulator calculates the discrepancies between the maximum and the two other smaller exponents. The results are fed into shift module to complete the mantissa alignment. Lastly, a tree adder with two levels is utilized to accumulate the aligned mantissa. A 3D output buffer, which is implemented using ping-pong FIFOs, is used to cache the temporary mantissa result from the channel accumulation submodule for successive accumulation.

3D Max Pooling — Pooling is performed on the pixels across the different frames and their BFP representations can be based on different exponents. The mantissas have to be aligned first which requires a similar architecture deployed in the accumulator. Once the number is aligned, the aligned mantissas can be compared directly and the maximum value is obtained and buffered in the off-chip memory as the inputs for the next convolution layer.

IV. Optimization Tool

This section describes a tool for optimizing parameters of the proposed 3D-CNNs hardware based on user constraints. The performance and accuracy of the FPGA implementation of 3D-CNNs using BFP arithmetic are affected by factors such as block size, bitwidth of the mantissa, single-precision and double-precision accumulation. Our tool contains three stages (Figure 9): Accuracy Prediction, Resource Modeling and Performance Optimization, to enable design space exploration to meet design requirements.

First, the Accuracy Prediction stage accepts a set of user constraints including accuracy requirements, and produces all the possible combinations of block sizes, and the corresponding mantissa bits, options about SP or DP accumulations. Since the block size determines both BRAM usage and accuracy, a

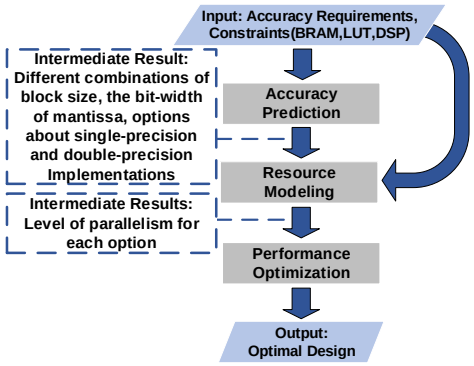


Fig. 9: Framework of the optimization tool.

BRAM model is used in calculating possible combinations of block sizes. Second, the Resource Modelling stage processes the combinations generated from the previous stage to produce the level of parallelism allowed for each combination, based on information about available resources from appropriate LUT and DSP models. Third, the Performance Optimization stage predicts optimized parameter values based on the results from the previous stages.

A. Accuracy Prediction and BRAM Modelling

Assume that the block size is B_s, the level of parallelism is P, the number of blocks is N_{block}, the bitwidth of the mantissa is L_m and the exponent bit is L_e. BRAM is utilized in the input and output buffers and its usage is given by:

$$BRAM = \frac{(B_s^2 \times N_c \times N_l) \times L_m + (N_{block} \times N_c \times N_l) \times L_e}{BRAM_{size}},$$

The $C3D$ tool [7] is a popular deep learning framework that implements 3D-CNNs in floating-point arithmetic, and it turns 3D convolution operations to matrix multiplications. To predict the accuracy of BFP-based 3D CNNs, we implement BFP-based 3D convolution by applying BFP-based matrix multiplications, block alignment and data conversion between BFP and floating point in $C3D$. Based on the BRAM model described above, all the possible combinations that satisfy the constraints can be calculated. Then, BFP-$C3D$ tests the accuracy of different combinations iteratively. Finally, only the combinations that satisfy the accuracy requirement is output from the Accuracy Prediction stage.

B. Resource Modelling with DSP and LUT Models

Since both the mantissa kernel and accumulator utilize DSP , the DSP usage is:

$$DSP = P \times [K_c^2 \times (K_c - 1) \times D_{mul} + K_c \times \frac{A_l \times (A_l + 1)}{2} \times D_{add}],$$

where D_{mul} and D_{add} are the DSP usage of multiplier and adder. A_l is the level of tree adder in one 2D kernel, which equals to $\lceil \log_2 (K_c^3) \rceil$. Both D_{mul} and D_{add} depend on the bitwidth of mantissa, as well as the selection of SP or DP implementations.

On the other hand, a linear regression model is utilized to approximate the consumption of LUT as it is difficult to predict. Its usage is given by:

$$LUT = \xi \times P,$$

where ξ is a linear function pre-trained based on different platforms with SP and DP implementations.

C. Performance Optimization

The execution time is based on the clock cycles of a 3D convolution layer which includes:

1) The computation time $T_{computation}$. Since the entire design is fully pipelined, the computation time is formulated as:
$$T_{computation} = \frac{W \times H \times N_f \times N_c \times N_l}{P},$$
2) Transfer time of the input data T_{in}. Despite the benefits of blocking and block alignment, the transfer of a block volume can still require certain cycles to finish. The transfer time is formulated as:
$$T_{in} = \frac{N_{in}}{Bandwidth \times Frequency},$$
where N_{in} is the amounts of input data.
3) Transfer time of the output data T_{out}, which is represented as:
$$T_{out} = \frac{N_{out}}{Bandwidth \times Frequency},$$
where N_{out} is the number of output data.

By evaluating the performance of every combination based on the above performance model, the combination with minimal execution time and corresponding parameters will be generated.

V. Experimental Results and Discussion

To recognize the performance and limitations of the proposed 3D-CNNs hardware for HAR acceleration, we implement the hardware design on Xilinx ZC706 platform which consists of a Kintex-7 FPGA and dual ARM Cortex-A9 processor. 1 GB DDR3 RAM is installed on the platform as off-chip memory. The hardware 3D-CNNs is clocked at 200 MHz while the ARM processor runs at 1 GHz. Vivado 2016.2 is used for synthesis and implementation. The Sports-1M dataset [8] which includes 1.1 million videos of 487 sports categories is used in the following experiments. The size of input tensor is 112×112, the channel number is 3, frame number is 16, and the batch size is 1.

A. Evaluation of the HAR Accuracy and Performance

Based on the descriptions in Section III and Section IV, the customization of the 3D-CNNs hardware is mainly a trade-off between performance and accuracy, as it is determined by the block size and also the selection of SP or DP accumulation. Therefore, we customize the FPGA design for two cases in this evaluation: maximum performance and highest accuracy, so as to determine the optimal implementation of the hardware 3D-CNNs.

Experiment I: Maximum Performance — By customizing the 3D-CNNs hardware for the highest performance, we can understand the lower bound of the HAR accuracy under BFP representation based on the given FPGA. This can also provide an understanding of the implications of SP and DP accumulation with regards to different precision.

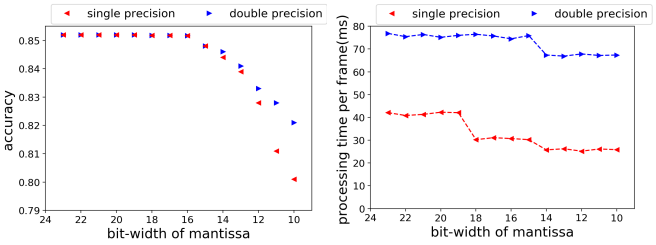
To achieve maximum performance, the block size of each layer is manually set to the maximal supported value, which is actually bounded by the BRAM resource due to blocking. Specifically, a larger block size increases the variance of the number within a block and subsequently aggravates the precision loss, but it also maximizes the utilization of BRAM and decreases the overhead of data transfer. With respect to ZC706 platform, the maximal supported block size from the 1st to 7th convolution layer is 28, 8, 7, 7, 7, 7 and 7.

Figure 10 shows the corresponding accuracy, performance and resource consumption using both SP and DP implementations. As displayed with the same trend on the left side of Figure 10a, SP implementation can surprisingly achieve an identical accuracy compared to DP implementation when the precision, i.e. the bitwidth of mantissa, is less than or equal to 15 bit. This is because the robustness of the C3D network can tolerance substantial precision loss before affecting the final HAR accuracy. Moreover, the performance of SP implementation is 2.6 times faster than that of DP implementation (Figure 10b) when the bitwidth of the mantissa is 15 bit. Finally, Figure 10c and Figure 10d show that the area cost for both implementations is bounded by the DSP resource. Given the same DSP budget, the decrease in precision does not decrease the DSP usage until it is less than certain bitwidth, i.e. 18 bits. With a reduced demand of DSP, the level of parallelism can be increased and this results in a slight increase in LUTs and registers usage in Figure 10c when the precision becomes 18 bit. This also contributes to an improved processing time for SP implementation when the precision is 18 bit.

Experiment II: Maximum Accuracy for HAR — Setting the 3D-CNNs hardware for the highest achievable accuracy can provide insights on the lower bound of the processing time based on the given FPGA. Since manual customization of the network based on accuracy requirements is non-trivial, we rely on the optimization tool to generate the relevant parameters for hardware customization, which can be used in evaluating the capabilities of the optimization tool.

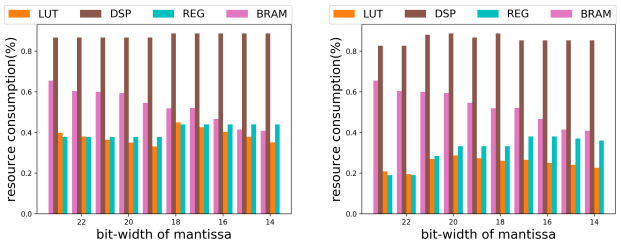
In this experiment, we fine-tune the accuracy requirement from 84.8% to the vicinity of maximum achievable accuracy, i.e. 85.2%, with an increment of 0.1% in each iteration. Each accuracy value is then supplied to the optimization tool to generate optimal parameters for the hardware network. Figure 11a shows the predicted runtime from the optimization tool and the actual runtime of the hardware. Table III displays different specifications obtained from the optimization based on different accuracy requirements.

The increase in the accuracy requirements from 84.8% to 85.1% results in an increase in the processing time by 0.07 to 1.4 times. We also obtain design parameters from the optimization tool with the accuracy requirements slowly



(a) HAR accuracy versus different precision.

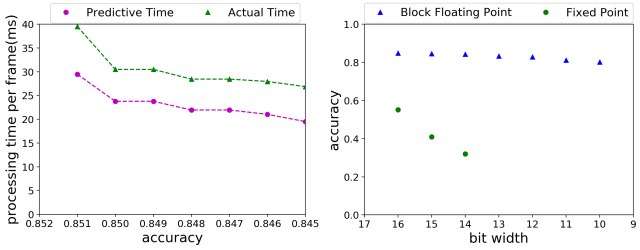
(b) Processing time per frame versus different precision.



(c) The resource consumption of SP implementation versus different precision.

(d) The resource consumption of DP implementation versus different precision.

Fig. 10: Evaluation of the HAR accuracy and performance with different precision based on the block size 28, 8, 7, 7, 7, 7 and 7 from the 1st to 7th convolution layer. Also displayed is the corresponding resource consumption for each precision.



(a)

(b)

Fig. 11: (a) The performance of the hardware 3D-CNNs versus accuracy based on the parameters produced by the optimization tool, (b) Accuracy comparison between fixed-point and BFP representation.

decreased from 84.8%. It can be seen that the processing time is not reduced significantly with the decrease of accuracy requirements. When the accuracy requirement is decreased from 84.8% to 84.5%, the processing time is only reduced by around 5%. Therefore, the implementation with the HAR accuracy 84.8% is considered to be the optimal design on the given FPGA and is therefore used as the final implementation.

B. Fixed-Point versus Block Floating-Point Representation

Most of the existing FPGA implementations of CNNs rely on fixed-point representation to achieve high performance. To quantitatively compare the accuracy discrepancies between fixed-point and BFP representations, the C3D tool is revised to implement 3D-CNNs based on fixed-point and BFP arithmetic.

TABLE III: Optimal design parameters provided by the optimization tool with different accuracy requirements.

Accuracy Requirement	Implementation	Precision/ Mantissa Bitwidth	Block Size
85.1%	Single-precision	21	28, 8, 7, 7, 7, 7, 7
85.0%	Single-precision	18	28, 8, 7, 7, 7, 7, 7
84.9%	Single-precision	16	28, 8, 7, 7, 7, 7, 7
84.8%	Single-precision	15	28, 8, 7, 7, 7, 7, 7
84.7%	Single-precision	15	28, 8, 7, 7, 7, 7, 7
84.6%	Single-precision	15	56, 8, 7, 7, 7, 7, 7
84.5%	Single-precision	15	56, 14, 14, 7, 7, 7

In this experiment, the block size is set to be 28, 8, 7, 7, 7, 7 and 7 from the 1st to 7th convolution layer and SP calculation is utilized in BFP implementation, which is known to be the optimal settings for performance based on the above evaluations. To ensure a fair comparison, the same block size is used and retraining is not applied in the fixed-point implementation. The result is presented in Figure 11b which shows that BFP can always outperform fixed-point representations in terms of accuracy regardless of the bitwidth. This clearly showcases the benefits of implementing 3D-CNNs with BFP representation on FPGA for HAR.

C. Performance Comparison with GPU and CPU

To compare the performance of the 3D-CNNs hardware on Xilinx ZC706 with other platforms, we use the final implementation that is customized for performance with SP accumulation and mantissa width = 15, exponent width = 8. Intel i7-950 (3.07 GHz) CPU and NVIDIA TITAN X GPU are used to run C3D as well. The compilation flag $-Ofast$ is activated and the rest of the settings are kept as the default ones. We also compare our final implementation to the previous work F-C3D [4] using the same FPGA device.

Table IV shows the performance, HAR accuracy and power consumption on different platforms. Our final design can achieve 33 fps with only 0.4% accuracy loss which is considered to be sufficient for HAR in many real-life applications. Moreover, our design outperforms the C3D tool on i7-950 CPU by 8.2 times and consumes 3.6 times less power than TITAN X GPU. Finally, the area cost of the final design based on Zynq ZC706 is shown in Table V.

TABLE V: Area cost of the final hardware on Zynq ZC706.

	LUT	Register	DSP48	BRAM
Available	218600	437200	900	545
Utilization	82849	195674	780	480
Percentage Used	37.9%	44.7%	86.6%	88.1%

VI. Conclusion and Future Work

This work proposes a novel FPGA design of 3D-CNNs that performs fast and accurate HAR (Human Action Recognition). The design makes use of block floating-point arithmetic, which provides a competitive classification accuracy and eliminates the need to retrain the network. Different optimization strategies such as locality exploration and block alignment with 3D blocking are developed, which improve both the performance and accuracy. An automatic tool is prototyped for determining the optimal parameters for network customization based on user constraints. Future work includes incorporating energy constraint into the optimization tool and applying Winograd algorithm for 3D CNNs to further improve the performance.

TABLE IV: Performance comparison of the final FPGA design versus CPU and GPU.

	CPU	GPU	F-C3D [4]	Our Work
Platform	Intel i7-950	TITAN X (Pascal)	Zynq ZC706	
No. of cores	8	3584	–	
Compiler	GCC 4.8.5	CUDA 8.0	Vivado 2016.2	
Flags	$-Ofast$		–	
Frequency	3.07 GHz	1.53 GHz	176 MHz	200 MHz
Precision	32 bit floating point		16 bit fixed point	sign: 1 bit, mantissa: 15 bit, exponent: 8 bit BFP
Technology	45 nm	16 nm	28 nm	
Processing Time per frame (ms)	243.75	4.79	33.3	29.8
Power (W)	130	168	9.7	9.9
Energy per frame (J)	31.68	0.8	0.32	0.29
Accuracy	85.2%	85.2%	55.1%	84.8%

Acknowledgement

The support of Corerain Technologies, Lee Family Scholarship, the EU Horizon 2020 Research and Innovation Programme under grant agreement number 671653 and the UK EPSRC (EP/L00058X/1, EP/L016796/1, EP/N031768/1 and EP/P010040/1) is gratefully acknowledged.

References

[1] X. Ma *et al.*, "Optimizing Hardware Design for Human Action Recognition," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–11.

[2] J. Shen *et al.*, "Towards a Uniform Template-based Architecture for Accelerating 2D and 3D CNNs on FPGA," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018, pp. 97–106.

[3] B. Jacob *et al.*, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," *arXiv:1711.02213*, 2017.

[4] H. Fan *et al.*, "F-C3D: FPGA-based 3-Dimensional Convolutional Neural Network," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–4.

[5] J. H. Wilkinson, *Rounding errors in algebraic processes*. Courier Corp., 1994.

[6] H. Meng *et al.*, "A Human Action Recognition System for Embedded Computer Vision Application," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, June 2007, pp. 1–6.

[7] D. Tran *et al.*, "Learning Spatiotemporal Features with 3D Convolutional Networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4489–4497.

[8] A. Karpathy *et al.*, "Large-scale Video Classification with Convolutional Neural Networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.

[9] E. Nurvitadhi *et al.*, "Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?" in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 5–14.

[10] K. Kalliojarvi and J. Astola, "Roundoff Errors in Block-floating-point Systems," *IEEE Transactions on Signal Processing*, vol. 44, no. 4, pp. 783–790, 1996.

[11] U. Köster *et al.*, "Flexpoint: An Adaptive Numerical Format for Efficient Training of Deep Neural Networks," *arXiv:1711.02213*, 2017.

[12] Z. Song *et al.*, "Computation Error Analysis of Block Floating Point Arithmetic Oriented Convolution Neural Network Accelerator Design," *arXiv:1709.07776*, 2017.

[13] Y. Jia *et al.*, "Caffe: Convolutional Architecture for Fast Feature Embedding," *arXiv:1408.5093*, 2014.

[14] U. Aydonat *et al.*, "An OpenCL Deep Learning Accelerator on Arria 10," in *2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 55–64.