F-E3D: FPGA-based Acceleration of an Efficient 3D Convolutional Neural Network for Human Action Recognition

Hongxiang Fan*, Cheng Luo†, Chenglong Zeng‡, Martin Ferianc*, Zhiqiang Que*, Shuanglong Liu*, Xinyu Niu§, Wayne Luk*

* Dept. of Computing, School of Engineering, Imperial College London, UK
{*h.fan17, m.ferianc15, z.que, s.liu13, w.luk*}*@imperial.ac.uk*
† State Key Laboratory of ASIC and System, Fudan University, Shanghai, China
‡ School of Microelectronics, Tianjin University, China
§ Corerain Technologies Ltd., Shenzhen, China, *xinyu.niu@corerain.com*

***Abstract*—Three-dimensional convolutional neural networks (3D CNNs) have demonstrated their outstanding classification accuracy for human action recognition (HAR). However, the large number of computations and parameters in 3D CNNs limits their deployability in real-life applications. To address this challenge, this paper adopts an algorithm-hardware co-design method by proposing an efficient 3D CNN building unit called 3D-1 bottleneck residual block (3D-1 BRB) at the algorithm level, and a corresponding FPGA-based hardware architecture called F-E3D at hardware level. Based on 3D-1 BRB, a novel 3D CNN model called *E3DNet* is developed, which achieves nearly 37 times reduction in model size and 5% improvement in accuracy compared to standard 3D CNNs on the UCF101 dataset. Together with several hardware optimizations, including 3D fused BRB, online blocking and kernel reuse, the proposed F-E3D is nearly 13 times faster than a previous FPGA design for 3D CNNs, with performance and accuracy comparable to other state-of-the-art 3D CNN models on GPU platforms while requiring only 7% of their energy consumption.**

I. Introduction

Over the past few years, human action recognition (HAR) for autonomous driving and intelligent video surveillance has become a popular research topic in computer vision and pattern recognition. Among various algorithms proposed for HAR, three-dimensional convolutional neural networks (3D CNNs) have demonstrated their outstanding classification accuracy [1], [2]. The success of 3D CNNs lies in the spatio-temporal 3D convolutional layers which incorporate the third-dimension information into the analysis. However, the algorithmic and space complexity of 3D convolution imposes a large overhead on the speed of 3D CNNs, which limits their deployment on real-life applications [3].

Different hardware devices such as FPGAs, ASICs and GPUs have been utilized to accelerate 3D CNNs. Among these hardware platforms, FPGAs are gaining popularity because of their better flexibility than ASICs and higher energy efficiency than GPUs [4]–[7]. However, it is challenging to design high-performance accelerators for 3D CNNs on FPGAs as the limited on-chip resources often cannot meet the heavy computational and memory demand [8]. Model compression is one kind of algorithm-level optimizations for improving the performance. Various compression techniques such as quantization and 3D Winograd algorithm have been proposed to reduce the algorithmic complexity of 3D CNNs [9], [10]. However, the limited compression rate still does not lead to significant speedup on FPGA platforms. To effectively compress 3D CNN models and improve their performance on FPGAs, we address this problem from another perspective: designing efficient 3D building units to replace the standard 3D convolutional layers in 3D CNNs.

Several efficient 2D CNN building units have been proposed to replace the standard 2D convolutional layers [11], [12]. Among these 2D CNN building units, the bottleneck residual block has demonstrated the best performance in both compression and accuracy [13]. Adopting an algorithm-hardware co-design method, this paper explores generalizing the bottleneck residual block to 3D CNNs at the algorithm level, and devising an associate FPGA-based architecture to accelerate the proposed 3D CNN building units at the hardware level.

However, there are several design challenges:

- The bottleneck residual block is originally designed for 2D computer vision tasks such as image classification and segmentation. To enable the capability of 3D data analysis, the bottleneck residual block needs to be extended to three dimensions. However, such 3D extension has different possibilities, and their accuracy is unexplored.
- The computation of the bottleneck residual block involves a large amount of intermediate data, which imposes a heavy overhead on memory usage. The situation may become worse when it is extended to three dimensions since 3D CNNs need to process more data than 2D CNNs.

To address the above challenges, we systematically analyse the different possibilities of 3D BRB, and propose the most efficient building layer called 3D-1 bottleneck residual block (3D-1 BRB) which has the capability of 3D data analysis. Due to the use of 3D depth-wise and point-wise convolution layers, 3D-1 BRB requires fewer number of computations

and parameters than thoes in the standard 3D convolutional layer. Based on 3D-1 BRB, an efficient 3D CNN model called *E3DNet* is developed, which achieves a high compression rate compared with the state-of-the-art 3D CNN models while maintaining the same level of accuracy. At the hardware level, a novel FPGA-based architecture, F-E3D, is proposed to accelerate *E3DNet*. Several innovative optimizations including 3D fused BRB, online blocking and kernel reuse are proposed to address the design issues introduced by limited memory and computational resources on FPGA platforms. The main contributions of this work are the following:

- An efficient 3D building block called 3D-1 bottleneck residual block (3D-1 BRB). Based on 3D-1 BRB, the proposed 3D CNN model, *E3DNet*, requires far fewer parameters than the state-of-the-art 3D CNN models while achieving the same level of accuracy (Section III).
- A novel hardware architecture called F-E3D, which accelerates different types of convolutional layers in 3D-1 BRB. Together with several optimizations including 3D fused BRB, online blocking and kernel reuse, the proposed FPGA-based accelerator can achieve real-time performance with high accuracy for the HAR task. (Section IV).
- Compared to the state-of-the-art 3D CNN models on GPU platforms, the proposed FPGA-based accelerator of *E3DNet* can achieve comparable performance and accuracy for human action recognition with higher energy efficiency (Section V).

II. Background

This section introduces the basic operation of 3D convolution. A brief description is then presented for depth-wise convolution and 2D bottleneck residual block (2D BRB). Table I summarizes the notation used in this paper.

TABLE I: Parameters used in 3D convolution and 2D BRB.

Parameter	Description
H	The height of input feature map
W	The width of input feature map
K_s	The spatial kernel size
K_t	The temporal kernel size
N_c	The number of channels
N_f	The number of filters
N_l	The length of frames
K_{dw}	The kernel size of depth-wise convolution
t	The expansion factor

A. 3D Convolution

To incorporate the information from different frames, 3D convolution computes features in both spatial and temporal dimensions. Figure 1 presents the basic operation of 3D convolution where the temporal kernel size K_t is 3. At the beginning, different 2D convolutional kernels are applied to different consecutive frames, which generates K_t frames of intermediate data. Then these intermediate results are accumulated together to produce one final output frame. Since the output frame contains the information from different input frames, 3D CNNs have the capability of integrating three dimensional information into the analysis.

TABLE II: The input and output tensors within 2D BRB.

Input Shape	Operation	Output Shape
$H \times W \times N_c$	Expansion Convolution	$H \times W \times N_c \times t$
$H \times W \times N_c \times t$	Depth-wise Convolution	$H \times W \times N_c \times t$
$H \times W \times N_c \times t$	Projection Convolution	$H \times W \times N_f$

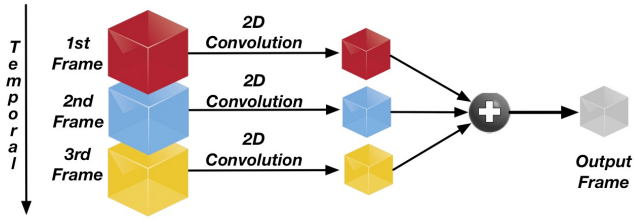

Fig. 1: The operation of 3D convolution.

B. Depth-wise Convolution and Bottleneck Residual Block

Depth-wise convolution is an efficient 2D convolutional layer in modern 2D CNNs. Figure 2 presents standard convolution versus depth-wise convolution. Compared to the standard convolution, depth-wise convolution only applies one single filter on each input feature map to generate output feature maps without channel accumulation, which significantly decreases the number of computations and parameters.

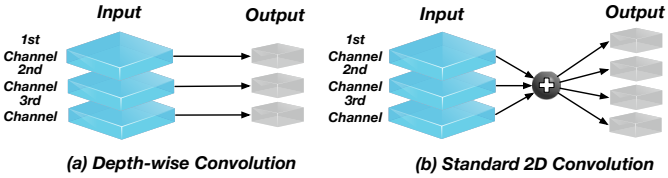

Fig. 2: The depth-wise and standard convolutions.

Based on the depth-wise convolution, the structure of 2D BRB is presented in Figure 3, where the expansion convolution and projection convolution are the standard convolutions with the kernel size being 1×1 (point-wise convolution). The batch normalization [14] (BN) layer is used after each convolution layer in 2D BRB, while the rectified linear unit with the threshold being 6 (ReLU6) is applied only after the expansion convolution and the depth-wise convolution. The shortcut addition is only active when the stride of depth-wise convolution is 1, which is utilized for residual learning [15]. Table II summarizes the input and output tensors of each convolution in 2D BRB. The 2D BRB has an expansion factor t, which is used to increase the internal dimension for better accuracy.

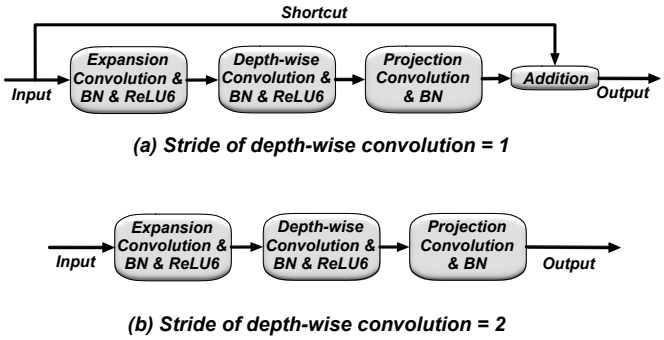

Fig. 3: The structure of 2D bottleneck residual block.

C. Related Work

The use of 3D convolution for extracting spatio-temporal features from videos is first proposed by Ji et al. [16]. Based on this work, *C3D* is then introduced by Tran et al. [17] for human action recognition, which has been adopted as a de facto standard for 3D CNNs. To demonstrate that 3D convolution has a better performance than 2D convolution in video understanding, Hara et al. [1] apply 3D convolution to the ResNet101 [15] backbone, and propose the state-of-the-art 3D CNN model called ResNeXt-101. However, the algorithmic complexity of 3D convolution imposes a large overhead on these networks, which limits their deployment for real-life applications [3]. Although Qiu et al. [2] propose a new network architecture called Pseudo-3D Residual Net (*P3D ResNet*) to decrease the number of computations and parameters, it does not achieve significant improvement on processing speed because of limited compression rate.

Various implementations have been proposed to accelerate 3D CNNs for inference on FPGAs. Shen et al. [10] developed an FPGA design to accelerate *C3D* based on a uniform template, where the 3D Winograd algorithm is used to decrease the arithmetic complexity of 3D convolution. Despite the high performance with 430.7 Gops on VC709, the accuracy of the 16-bit fixed-point accelerator is unexplored, and its applicability to real-life situations is unclear. A novel customizable architecture based on block floating-point (BFP) arithmetic is proposed by Fan et al. [9], which significantly reduces resource utilization on FPGA devices. However, it still cannot compete with GPU implementations in terms of processing speed.

III. E3DNet

A. 3D CNNs Building Block

Inspired by the 2D bottleneck residual block (2D BRB), a similar three-layer bottleneck structure can be developed as a building block of 3D CNNs. We first extend the second layer, 2D depth-wise convolution, to three dimensions. Figure 4 presents the basic operation of 3D depth-wise convolution where N_c=2 and K_t=3. At the beginning, the input volume is separated by the channel dimension, which generates N_c sets of data with size $K_t \times H \times W$. Then different 2D depth-wise convolutions (2D-DW Convs) are applied to N_c groups of data separately, where each group generates K_t numbers of output feature maps. The results within one group are accumulated in a final output frame.

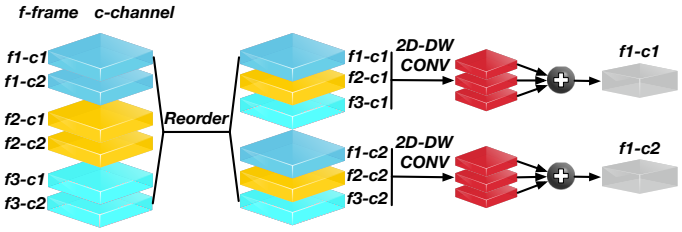

Fig. 4: 3D depth-wise convolution with N_c=2 and K_t=3.

For the first and third layers, we use the same spatial kernel settings as those for 2D BRB. In temporal dimension, to enable the capability of 3D data analysis, a temporal kernel with K_t size is applied to one of these two layers. Therefore we propose two variants of building block for 3D CNNs, 3D-1 BRB and 3D-3 BRB, where the number indicates the position of the 3D convolution. Figure 5 shows the structure of these two variants. Note that the batch normalization (BN) layer is used after all convolutional layers, while the rectified linear unit (ReLU) is only activated after the first two layers.

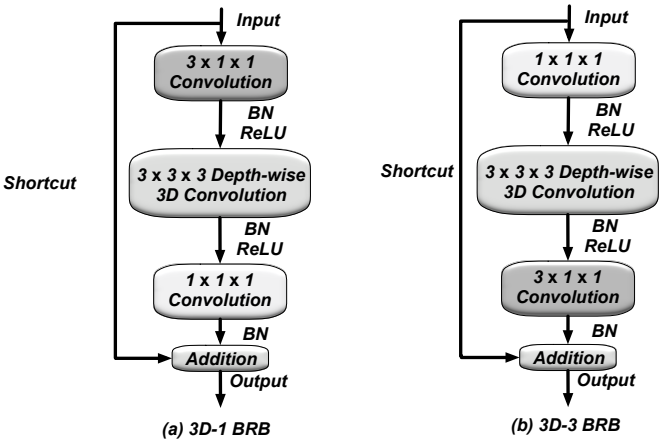

Fig. 5: Two 3D bottleneck residual blocks, with different position of the 3D (3x1x1) convolution.

To find the building block which is more efficient, we first compare the computational cost of these two variants. According to the parameters in Table I, the number of multiply-add (MAdd) operations required by these two variants is shown in Table III.

TABLE III: The number of multiply-add (MAdd) operations in two variants of 3D BRB.

	Operations
$M\text{-}Adds_{3D-1}$	$H \times W \times N_c \times t \times (K_t \times N_c + K_t \times K_{dw} + N_f)$
$M\text{-}Adds_{3D-3}$	$H \times W \times N_c \times t \times (N_c + K_t \times K_{dw} + K_t \times N_f)$

Comparing the computational load of 3D-1 and 3D-3 BRBs, we subtract $M\text{-}Adds_{3D-3}$ from $M\text{-}Adds_{3D-1}$ to obtain:

$$H \times W \times N_c \times t \times (K_t - 1) \times (N_f - N_c). \quad (1)$$

As N_f is greater than N_c and K_t is bigger than one, Equation (1) is larger than zero, which means $MAdds_{3D-3}$ is greater than $MAdds_{3D-1}$. Therefore the 3D-3 BRB requires more computations than 3D-1 BRB. Because our target platform has limited computational resources, 3D-1 BRB is selected as the 3D CNN building block in this paper.

Another benefit of 3D-1 BRB is that it requires fewer amount of on-chip memory than 3D-3 BRB. For simplicity, we only compare the memory cost introduced by the first and third layers because both 3D-1 and 3D-3 BRBs have the same 3D depth-wise convolution in the middle. Table IV summarizes the memory usage of the first and third layers in 3D-1 and 3D-3 BRBs. As 3D convolution needs to process K_t consecutive frames simultaneously, K_t number of input frames are required to cache in the on-chip memory. Therefore, K_t is applied to the memory cost of the first layer in 3D-1 BRB, and the third layer in 3D-3 BRB. Since K_t=3 and the expansion factor t=6, we observe that the memory usage of 3D-1 BRB is less than that of 3D-3 BRB. Therefore the selection of 3D-1 BRB can also reduce the amount of on-chip memory.

TABLE IV: The memory cost of the first and third layers.

Memory Cost	3D-1 BRB	3D-3 BRB
1st Layer	$H \times W \times N_c \times K_t$	$H \times W \times N_c \times 1$
3rd Layer	$H \times W \times (N_c \times t) \times 1$	$H \times W \times (N_c \times t) \times K_t$
Total	$H \times W \times N_c \times (K_t + t)$	$H \times W \times N_c \times (t \times K_t + 1)$

B. Network structure of E3DNet

The detailed structure of the 3D-1 BRB is shown in Table V. From the top to the bottom, we call these three layers $3 \times 1 \times 1$ temporal convolution (TMP Conv), $3 \times 3 \times 3$ 3D depth-wise convolution (3D DW Conv), and $1 \times 1 \times 1$ point-wise convolution (PW Conv). There is a shortcut when the stride of temporal convolution is $1 \times 1 \times 1$, which keeps the consistency with 2D BRB for residual learning.

TABLE V: The detailed structure of 3D-1 BRB.

Input Tensor	Convolution	Output Tensor
$H \times W \times N_c \times N_l$	TMP Conv, ReLU	$H \times W \times N_c \times N_l \times t$
$H \times W \times N_c \times N_l \times t$	3D DW Conv, ReLU	$H \times W \times N_c \times N_l \times t$
$H \times W \times N_c \times N_l \times t$	PW Conv	$H \times W \times N_f \times N_l$

Based on the 3D-1 BRB, an efficient 3D CNN model called *E3DNet* is proposed. The network starts from one $1 \times 3 \times 3$ convolution with 45 filters and $1 \times 2 \times 2$ stride, followed by a $3 \times 1 \times 1$ temporal convolution. *E3DNet* uses 19 3D-1 BRBs with the global average pooling (GAP) and point-wise convolution at the bottom. Table VI shows the architecture of *E3DNet*, where each line describes a sequence of identical building blocks with expansion factor t and repeated number n. The stride of the first block in each sequence is s, and all others are $1 \times 1 \times 1$. k is the number of classes for the classification tasks, which can vary for different datasets.

TABLE VI: The network architecture of *E3DNet*.

Input	Operation	t	N_f	n	s
$16 \times 112^2 \times 3$	Conv $1 \times 3 \times 3$	-	45	1	$1 \times 2 \times 2$
$16 \times 56^2 \times 45$	Conv $3 \times 1 \times 1$	-	64	1	$1 \times 1 \times 1$
$16 \times 56^2 \times 64$	3D-1 BRB	1	24	1	$1 \times 1 \times 1$
$16 \times 56^2 \times 24$	3D-1 BRB	6	24	2	$1 \times 1 \times 1$
$16 \times 56^2 \times 24$	3D-1 BRB	6	48	4	$2 \times 2 \times 2$
$8 \times 28^2 \times 48$	3D-1 BRB	6	64	6	$2 \times 2 \times 2$
$4 \times 14^2 \times 64$	3D-1 BRB	6	96	3	$2 \times 2 \times 2$
$2 \times 7^2 \times 96$	3D-1 BRB	6	512	1	$1 \times 1 \times 1$
$2 \times 7^2 \times 512$	GAP	-	-	1	$1 \times 1 \times 1$
$1 \times 1^2 \times 512$	Conv $1 \times 1 \times 1$	-	k	1	$1 \times 1 \times 1$

IV. FPGA Accelerator Design

A. Design Methodology

To improve the scalability of the design, our approach adopts a single processing engine architecture [18] devised to process one layer or one block at a time, and the whole network is computed by repeatedly configuring different layers of the same design.

1) Fused 3D BRB: We adopt the roofline model [19] to estimate the performance limitation of the proposed 3D-1 BRB. The model contains one roofline curve characterizing the theoretical peak performance provided by the target device, and several vertical lines representing the algorithm intensity of different operations. The interaction between the roofline curve and each algorithm-intensity line presents the theoretical peak performance of each operation, which is either bounded by computational resources or by memory bandwidth.

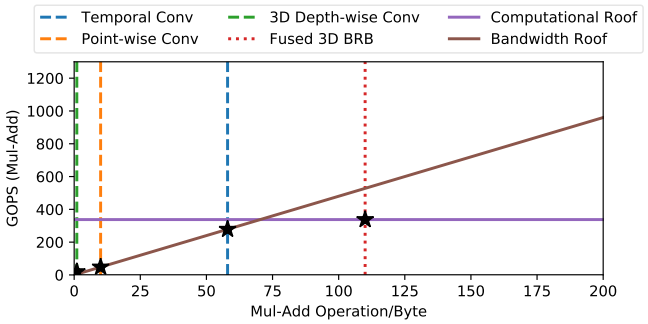

Fig. 6: Roofline model of *E3D*.

Figure 6 shows our roofline model targeting the Intel Arria 10 SX660 FPGA board, with 200 MHz clock frequency and 4.8 GB/s of DRAM bandwidth.[1] It is clear that all the temporal, 3D depth-wise and point-wise convolutions are memory-bound operations, and hence the overall performance will also be bounded by the memory bandwidth. To solve

[1]Assuming that the operation refers to single-precision floating-point multiply-add (Mul-Add), only DSP resource is utilized to perform the computation and each Mul-Add costs 1 DSPs.

this problem, we propose the fused 3D BRB: instead of accelerating these three convolutional layers separately, the fused 3D BRB takes the 3D-1 BRB as the basic acceleration module, where all the intermediate data within the 3D-1 BRB are cached in on-chip memory for data reuse. After the fused 3D BRB optimization, the overall performance of 3D-1 BRB becomes compute-bound, which is given by the dotted red line in Figure 6.

2) Online Blocking: An issue which comes with the 3D fused BRB is that it requires a large amount of on-chip memory to cache all the intermediate results within the 3D-1 BRB. To solve this problem, we propose a novel blocking strategy called online blocking to reduce on-chip memory usage by changing the computational sequence of 3D-1 BRB.

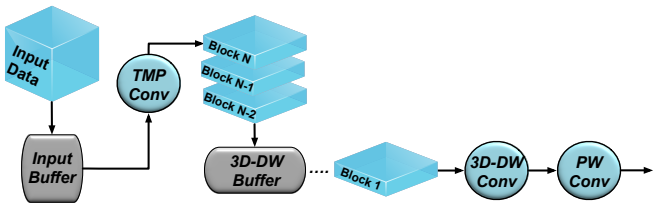

Fig. 7: The process of online blocking.

Figure 7 illustrates the process of our proposed online blocking strategy. Thanks to the structure of the bottleneck residual block, the channel number of temporal convolution is far smaller than the standard 3D Convolution. Therefore, all the input data of the temporal convolution can be cached in on-chip memory for data reuse. Then instead of calculating all the results required by the 3D depth-wise convolution (3D DW CONV), the temporal convolution (TMP CONV) only computes $\frac{H\times W}{B_{num}} \times N_f \times K_t$ elements of the results, where B_{num} is the number of blocks. These intermediate data will be immediately processed by the 3D depth-wise and point-wise convolutions (PW CONV), and then transferred to off-chip memory. The whole computation of 3D-1 BRB consists of repeating this computational pattern. Since the on-chip memory only needs to cache $\frac{H\times W}{B_{num}} \times N_f \times K_t$ elements of intermediate data, the memory usage can be significantly reduced. Algorithm 1 illustrates the pseudocode of 3D-1 BRB with the online blocking strategy.

Algorithm 1 3D-1 BRB with Online Blocking.

```
for frm = 0 to N_l do
    for blk = 0 to B_num do
        for f_st = frm to frm + K_t do
            for fltr = 0 to N_f do
                for c = 0 to N_c do
                    for h = blk×H/B_num to (blk+1)×H/B_num do
                        for w = 0 to W do
                            outpt[frm][blk][fltr][h][w]+=
                            tmp_conv(inpt[f_st][blk][c][h][w]);
    dw_outpt[frm][blk] = 3d_dw_conv(outpt[frm][blk])
    pw_outpt[frm][blk] = pw_conv(dw_outpt[frm][blk])
```

3) Kernel Reuse: To improve hardware efficiency, the computational resources allocated for these three convolutional layers should be proportional to their number of computations. However, since the computation and corresponding proportions of the three convolutional layers vary in different 3D-1 BRBs, deploying computational kernels for each convolution may cause hardware inefficiency. To solve this problem, we propose a kernel reuse strategy to improve hardware efficiency by reusing the computational kernel of 3D depth-wise convolution with the temporal and point-wise convolutions.

The computational kernel deployed on hardware for 3D depth-wise convolution (computational engine in Section IV-B1) consists of $K_t \times K_{dw}^2$ multipliers followed by a $\lceil \log_2 (K_t \times K_{dw}^2) \rceil$-level adder tree, where both K_t and K_{dw} equal to 3. As the kernel size of 3D depth-wise is different from those of the temporal and point-wise convolutions, the computational kernel cannot directly be used by other convolutions. To address this challenge, this paper applies loop unrolling and loop interchanging to map the computation of temporal and point-wise convolutions into the computational kernel of 3D depth-wise convolution.

Algorithm 2 Kernel Reuse with Temporal Convolution.

```
for frm = 0 to N_l do
    for f_th = frm to frm + K_t do                          ▷ Loop Interchange
    for fltr = 0 to N_f do
        for c = 0 to N_c do                                 ▷ Loop Unrolling
        for c = 0 to N_c/(K_dw×K_dw) do
            for h = 0 to H do
                for w = 0 to W do
                    for c_unrol = c × K_dw^2 to (c+1) × K_dw^2 do
                        for f_th = frm to frm + K_t do
                            outpt[frm][fltr][h][w]+=
                            coef[f_th][fltr][c_unrol] ×
                            inpt[f_th][c_unrol][h][w];
```

Algorithm 2 illustrates how the temporal convolution is mapped into the computational kernel of 3D depth-wise convolution. First, the loop with variable f_th is interchanged into the innermost loop (red box). Second, loop unrolling is applied to the loop with variable c to separate the channel accumulation into several groups, where each group contains K_{dw}^2 multiplication-additions (blue box). After these two loop changes, the innermost two loops (blue and red boxes) containing $K_t \times K_{dw}^2$ multiplication-additions has the same kernel size as that in 3D depth-wise convolution. Therefore, the calculation of temporal convolution can be mapped and computed in the computational kernel of 3D depth-wise convolution.

Similar optimizations can be applied to the point-wise convolution, which separates the channel accumulation into several sets of $K_t \times K_{dw}^2$ multiplication-additions to fit the calculation into the computational kernel of 3D depth-wise convolution.

B. Hardware Design

1) Hardware Architecture: Based on the design methodology introduced above, the hardware architecture F-E3D is presented in Figure 8, which mainly consists of a computational engine, sliding window, ReLU, pooling modules and several buffers. The matrix buffer is used to cache the input pixels

required by the computational engine. The sliding window receives the data one by one and outputs $K_t \times K_{dw} \times K_{dw}$ pixel matrix into the computation engine for the calculation of 3D depth-wise convolution.

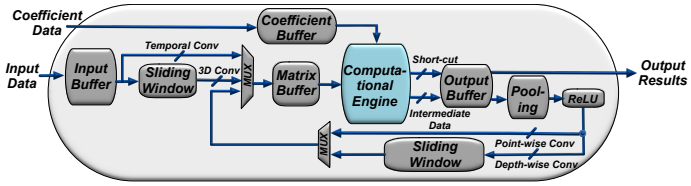

Fig. 8: Hardware architecture of F-E3D.

Computational Engine — The basic element in computational engine is the computational unit (CU), which is shown in Figure 9. Each CU consists of K_t processing elements (PEs) followed by a $\lceil \log_2 (K_t) \rceil$-level adder tree. The basic component of PE is the processing unit (PU), which is composed of K_{dw} multipliers and a adder tree with $\lceil \log_2 (K_{dw}) \rceil$ levels. Both K_t and K_{dw} equal to 3.

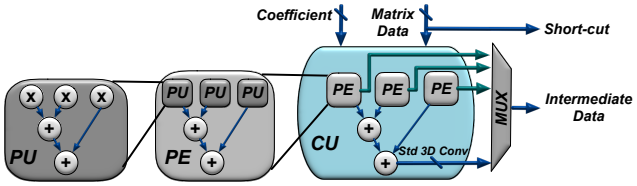

Fig. 9: Design of computational unit.

When the computational engine is used for 3D depth-wise convolution, the adder tree in each CU is enabled and the outputs come from the CU through the last two-level adder tree. As for other convolutions, the results are output from the PE directly.

Output Buffer — Figure 10 presents the design of the output buffer. Three buffers are utilized to cache the intermediate data of point-wise, 3D depth-wise and temporal convolutions respectively. There is an accumulator module designed for channel accumulation, which is disabled for depth-wise convolution.

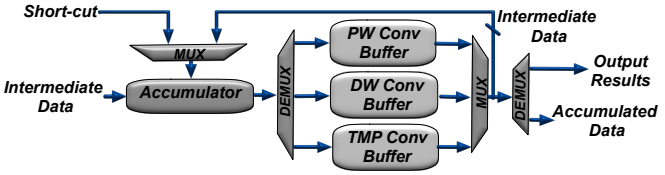

Fig. 10: Implementation of output buffer.

2) Computational Flow: The computation starts from the temporal convolution. All the input data are cached in the input buffer module for data reuse. Then the inputs are streamed into the computational engine through the matrix buffer, where intermediate results are accumulated in the output buffer. Because of the online blocking strategy, the computation of temporal convolution stops when one block of results is ready. That block of data is then fed into the ReLU, sliding window and computational engine for 3D depth-wise convolution. The process of point-wise convolution starts when 3D depth-wise convolution finishes the computation of one block. After that, the outputs of point-wise convolution are transferred back to the off-chip memory.

V. Experimental Results and Discussion

The whole design is implemented on an Intel Arria 10 SX660 platform using Verilog. The hardware F-E3D is clocked at 150 MHz. 1GB DDR4 SDRAM is installed on the platform as off-chip memory. Quartus 17 Prime Pro is used for synthesis and implementation. The UCF101 dataset [20] which includes 13320 videos of 101 human action categories is used in the following experiments for HAR.

A. Implementation Detail

Figure 11 shows the implementation detail of the proposed design. The avalon memory mapped interface (Avalon-MM) is used for the interaction between different components. The input data are transferred onto the FPGA through the peripheral component interconnect express 3.0 (PCIe3). DMA is used to access the coefficient data and intermediate results on DDR4. The CNN engine contains 176 CUs for parallel processing. Since the design is based on the single processing engine architecture, multiple 3D-1 BRBs are executed on the same FPGA design, where the configuration parameters are specified through the APB bus before the computation.

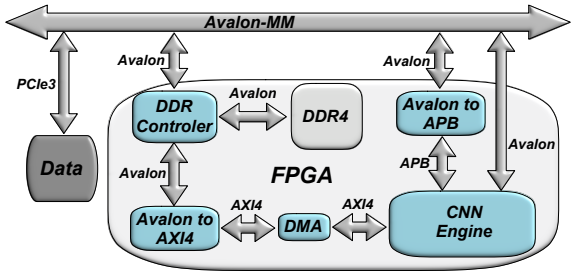

Fig. 11: Implementation detail of the design.

B. Model Size, Computatinal Cost and Accuracy

To demonstrate the efficiency and accuracy of 3D-1 BRB, the proposed 3D CNN model, *E3DNet*, is trained and tested on UCF101 dataset for HAR.[2] Video frames are scaled to the size of 128×171 and then each frame is generated by randomly cropping windows of size 112×112. The *E3DNet* takes one video clip as input, where each video clip contains L consecutive frames randomly sampled from the video with temporal jittering. Although large L may increase the accuracy, it also imposes a heavier computational load on the network. Therefore, L is set to be 16 in this paper as a trade-off between speed and accuracy. The *E3DNet* is first pretrained on

[2]The evaluation codes and model of our *E3DNet* are publicly available at: https://github.com/os-hxfan/E3DNet.git

the Kinetics [21] dataset, and then finetuned on the UCF101 dataset [20]. The accuracy reported in this paper is the Clip@1 accuracy where the prediction uses only one clip per video.

TABLE VII: The model size, computatinal cost and accuracy of different 3D CNN models published on top computer vision conferences.

	E3DNet	*ResNeXt -101* [1]	*P3D* [2]	*C3D* [17]
Clip@1 Accuracy	85.17%	87.7%	84.2%	79.87%
Model Size	8.6MB	365MB	261MB	321MB
Compression Rate of Model	1	42.3	30.3	37.3
MAdds	6.1G	9.8G	19.2G	38.2G
Compression Rate of MAdds	1	1.6	3.1	6.2

Table VII compares the proposed *E3DNet* with other state-of-the-art 3D-CNN models in terms of model size, computational cost and accuracy on the UCF101 dataset, which is shown in . All these models use 16 frames in one input video clip. Note that, although some models can achieve higher accuracy with support vector machine (SVM), we do not use SVM in this experiment for a fair comparison. Our *E3DNet* is nearly 37 times smaller and 5% more accurate than the classic 3D CNN model *C3D*. Compared with *P3D*, the proposed *E3DNet* improves the accuracy by 0.97% with approximately 30 times fewer parameters. Although the accuracy of *ResNeXt-101* is 2.53% higher than that of *E3DNet*, it also contains nearly 42 times more parameters. Therefore, when targeting platforms with limited resources, *E3DNet* is the most suitable 3D CNN model for human action recognition due to its high efficiency on parameters and computations.

C. Performance Comparison

To compare the performance of the proposed design on Intel Arria 10 SX660 with other platforms, we implement the *E3DNet* on Intel Xeon E5-2680 v2 CPU (20 cores) and NVIDIA TITAN X Pascal GPU (3584 cores) based on the MXNet framework [22]. The MKLDNN library is used for optimizing the CPU implementation. The GPU solution is optimized with the TensorRT and CuDNN 7.41 libraries. We also compare our final implementation to the previous FPGA-based accelerators [9] for *C3D*.

Table VIII shows the performance and power consumption of different platforms. The GPU implementations are evaluated in both 1 and 64 batches. Although the 64-batch implementation can increase the number of processed clips per second, it also introduces larger overhead on latency, which is not suitable for human action recognition with the requirement of real-time processing. Therefore, all the implementations are compared based on one batch. Among one-batch designs, our FPGA design is 196 and 3.4 times faster, and consumes 733 and 51 times less power than the *E3DNet* on the Xeon E5-2680 v2 CPU and the TITAN X Pascal GPU respectively. Note

TABLE VIII: Performance comparison of our final FPGA design versus CPU, GPU and other FPGA designs.

	CPU	**GPU**		**FPGA [9]**	**Our Work**
Platform	Intel Xeon E5-2680 v2	TITAN X Pascal		Xilinx ZC706	Intel Arria 10 SX660
Frequency	2.8 GHz	1.53 GHz		200 MHz	150 MHz
Technology	22 nm	16 nm		28 nm	20 nm
Model	*E3DNet*	*E3DNet*		*C3D*+ SVM	*E3DNet*
No. of batches	1	1	64	1	1
Acceleration Library	MKLDNN	TensorRT and CuDNN 7.41 libraries		-	-
Power (W)	135	144	240	9.9	36
Precision	32bit-float	32bit-float	32bit-float	block-float	32bit-float
Accuracy	85.17%	85.17%	85.17%	$\leq$ 81.99%	85.17%
Latency per clip (ms)	6921.3	121.5	41.1	476.8	35.3
Clips per second (cps)	< 1	8.23	24.3	2.1	29.3
Energy per clip (J)	931.5	17.49	9.86	4.72	1.27

that, even with the acceleration libraries including MKLDNN, TensorRT and CuDNN, the speed of *E3DNet* on GPU and CPU is still far less than the theoretical performance of CPU and GPU platforms. The potential reason may be that the 3D depth-wise convolution cannot achieve high performance on GPU and CPU, where the 2D depth-wise convolution has the similar performance issue [23]. Compared to the state-of-the-art FPGA-design [9] for *C3D* with SVM on Xilinx ZC706, our implementation is nearly 13 times faster with nearly 3% improvement in accuracy. Finally, the area cost of the final design based on Arria 10 SX660 is shown in Table IX.

TABLE IX: Area cost of the final hardware on Intel Arria 10 SX660.

	ALMs	**DSPs**	**M20K**
Available	251680	1687	2133
Utilization	113828	1584	1578
Percentage Used	45.2%	93.3%	74%

Figure 12 compares the proposed FPGA-based accelerator of *E3DNet* with other state-of-the-art 3D CNN models on GPU platforms, where the size of each dot is proportional to their power consumption. Since the input comes from a camera clip by clip in real-life human action recognition, all these implementations are evaluated in one batch. Note that the settings of GPU and FPGA are consistent with the configuration shown in Table VIII, and cudnn and TensorRT libraries are used to optimize all the GPU implementations. It is clear that the proposed FPGA implementation of *E3DNet* is slightly faster than the *P3D* and *ResNetXt-101* on GPU with nearly the same level of accuracy. Although *C3D* has the better performance in latency, it is nearly 5% less accurate than *E3DNet*. With regards to energy efficiency, the power

consumption of *E3DNet* on FPGA (36 W) is at least three times less than that of *C3D*, *P3D* and *ResNetXt-101* on GPU (170W, 120W and 125W).

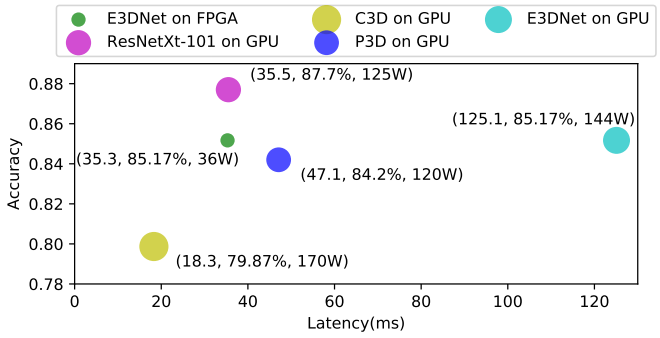

Fig. 12: Comparison of *E3DNet* on FPGA versus the state-of-the-art 3D CNNs on GPU. The size of the dot is proportional to the power consumption.

VI. Conclusion and Future Work

This paper adopts an algorithm-hardware co-design method for the acceleration of human action recognition. At the algorithm level, a novel 3D CNNs building unit called 3D-1 bottleneck residual block (3D-1 BRB) is first proposed. Based on 3D-1 BRB, an efficient 3D CNN model called *E3DNet* is developed, which is nearly 5% more accurate and 37 times smaller than the de facto standard 3D CNN. At the hardware level, an FPGA-based hardware architecture, F-E3D, is designed for *E3DNet*. Together with several optimizations includings 3D fused BRB, online blocking and kernel reuse, the proposed FPGA-based accelerator for *E3DNet* can achieve nearly 13 times speedup than a previous FPGA-based 3D CNNs accelerator. Future work includes improving the accuracy of *E3DNet* for human action recognition, exploring the 3D-1 BRB for other 3D computer vision tasks, optimizing the performance of 3D-1 BRB on GPU and CPU platforms.

Acknowledgement

The support of the China Scholarship Council, United Kingdom EPSRC (grant numbers EP/L016796/1, EP/N031768/1, EP/P010040/1 and EP/L00058X/1), Corerain, Maxeler, Intel and Xilinx is gratefully acknowledged.

References

[1] K. Hara, H. Kataoka, and Y. Satoh, "Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 18–22.

[2] Z. Qiu, T. Yao, and T. Mei, "Learning Spatio-temporal Representation with Pseudo-3D Residual Networks," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 5534–5542.

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

[4] 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.

[5] R. Zhao, S. Liu, H.-C. Ng, E. Wang, J. J. Davis, X. Niu, X. Wang, H. Shi, G. A. Constantinides, P. Y. Cheung *et al.*, "Hardware Compilation of Deep Neural Networks: An Overview," in *IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2018, pp. 1–8.

[6] S. Liu, X. Niu, H. Fan, 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)*, 2018.

[7] S. Liu, G. Mingas, and C.-S. Bouganis, "An unbiased mcmc fpga-based accelerator in the land of custom precision arithmetic," *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 745–758, 2017.

[8] 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 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 75–84.

[9] 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 *28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 287–294.

[10] 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 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2018, pp. 97–106.

[11] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.

[12] F. Chollet *et al.*, "Xception: Deep Learning with Depthwise Separable Convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1800–1807.

[13] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[14] S. Ioffe *et al.*, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv preprint arXiv:1502.03167*, 2015.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[16] 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, 2013.

[17] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning Spatiotemporal Features with 3D Convolutional Networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4489–4497.

[18] J. Misra and I. Saha, "Artificial Neural Networks in Hardware: A Survey of Two Decades of Progress," *Neurocomputing*, vol. 74, no. 1-3, pp. 239–255, 2010.

[19] S. Muralidharan *et al.*, "A Semi-Automated Tool Flow for Roofline Anaylsis of OpenCL Kernels on Accelerators," in *Proc. 1st Intl. Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC15)*, 2015.

[20] K. Soomro and M. Shah, "UCF101: A Dataset of 101 Human Actions Classes from Videos in the Wild," *arXiv preprint arXiv:1212.0402*, 2012.

[21] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev *et al.*, "The Kinetics Human Action Video Dataset," *arXiv preprint arXiv:1705.06950*, 2017.

[22] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems," *arXiv preprint arXiv:1512.01274*, 2015.

[23] R. Zhao, H.-C. Ng, W. Luk, and X. Niu, "Towards Efficient Convolutional Neural Network for Domain-Specific Applications on FPGA," in *28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 147–1477.