**SECTION II**

**OVERVIEW OF CONVOLUTION**

**Standard Convolution:**

A standard convolutional layer primarily comprises two pivotal operations: local correlation and receptive field traversal. In this context, each kernel serves as a filter, extracting localized features. During the forward propagation phase, the filter computes the nodes in the output feature maps by convolving the input feature maps with the kernel matrix. Fig. 1. illustrates the standard convolution. Consider a scenario where the current layer possesses input channels and output channels. The Standard Convolution operation takes the input feature map of dimensions and generates the output feature map of dimensions , where and represent the spatial widths of and respectively, and and denote the spatial heights of and respectively. The standard convolution is parameterized by the convolution kernel *K* of dimensions , where and represent the width and height of the kernel respectively.

The relation between , ,,, , with unit stride is given by eq. 1(a) and 1(b).

1(a)

1(b)

The expression for the Output feature map *O* of the Standard convolution is given by:

*(i , j)* represent the index of the output node, *x* is the index of the input channel, *n* is the index of the output channel.

The computational cost of standard convolution is .

**Depth-wise Separable Convolution:**

The standard convolution often entails high computational costs due to its extensive parameter requirements and computations, limiting its applicability in resource constrained environments. Conversely, Depth-wise Separable Convolution offers a solution by deconstructing the convolution process into Depth-wise and Point-wise convolutions. This decomposition drastically diminishes the parameter count and computational workload, consequently enhancing computational efficiency without compromising model performance. Fig. 2. Illustrates the Depth-wise separable convolution.

**(i) Depth-wise Convolution:**

Depth-wise convolution excels in capturing channel-specific features adeptly. In contrast to Standard Convolution, which employs a singular filter across all input channels, depth-wise convolution conducts convolution operations on each input channel autonomously, utilizing a distinct single-channel filter for each.

The expression for the intermediate output feature map *I’* of the Depth-wise convolution is as follows:

*I* represent input feature map, *KD* represent Depth-wise Convolution filter and *a* represents input channel index and *a’* represent filter index.

The computational cost of the Depth-wise convolution is .

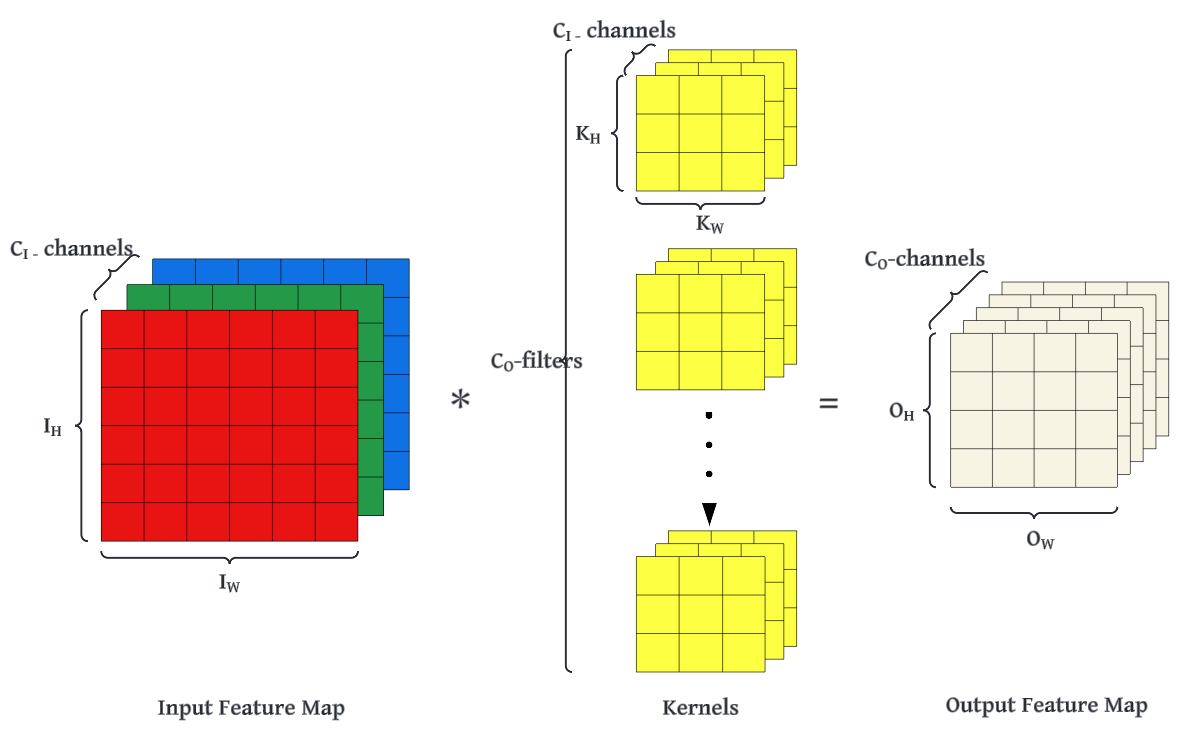


Fig.1. Standard Convolution

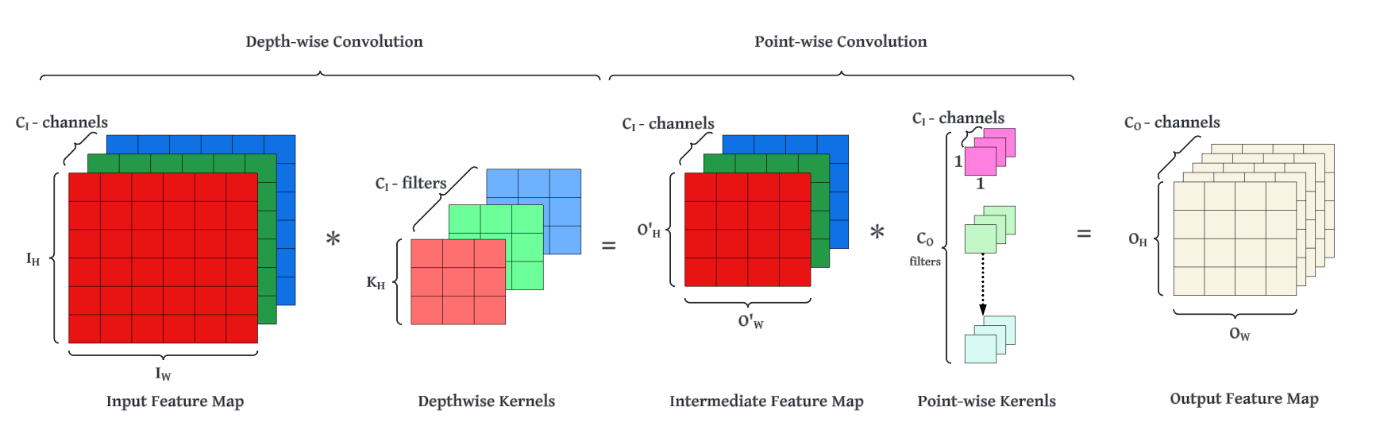


Fig.2. Depth-wise Separable convolution

**Algorithm 1: Depth-wise Convolution**

***Input****: I (IW\*IH) <= Input feature map*

*KD (Kw\*KH) <= Depth-wise Kernel*

***Output****: I’(OW\*OH) <= Output feature Map*

***Begin***

***SET*** *O= 0;*

*for i = 1 to OW do*

*for j=1 to OH do*

*for x=1 to Kw do*

*for y=1 to KH do*

*I’[i][j] += I[i+x-1] [j+y-1] \* KD[x][y];*

*end for*

*end for*

*end for*

*end for*

***End***

**(ii)Point-wise Convolution:**

Point-wise Convolution is the standard convolution where the intermediate results of the Depth-wise Convolution are convolved with 1\*1 filters with *CI* number of channels.

Let *I’* be the intermediate input feature map of size *OH’ \* OW’\* CI (such that OH = OH’ and OW =OW’)*and *KP* be the pointwise kernel of size *1 \* 1 \* CI \* CO* where *CO* is the desired number of output channels.

The output feature map O is expressed as

where O(i,j,k) is value of the output feature map at position *(i, j)* and channel *k*.

I’i,j,c is the value of intermediate input feature map at position *(i, j)* and channel *c*.

Kp1,1,c,k is the value of kernel at position (1,1) for input channel *c* and output channel *k*.

This expression represents the linear combination of input channels at each pixel location which is the essence of point-wise convolution.

The computational cost of the Pointwise convolution is

The overall computational cost of the Depth-wise Separable Convolution is

+

By expressing convolution as two step process of filtering and combining, we get a reduction in computation of:

**Algorithm 2: Point-wise Convolution**

***Input****: I’ (OW \* OH \* CI) <= Intermediate input feature map*

*KP (1 \* 1 \* CI \* CO) <= Point-wise Kernel*

***Outpu****t: O(OW\*OH\*CO) <= Output feature Map*

***Begin***

***SET*** *OF = 0;*

*for i = 1 to OW do*

*for j=1 to OH do*

*for x=1 to CO do*

*for y=1 to CI do*

*O[i][j][x] += I’[i][j][y] \*KP[x][y];*

*end for*

*end for*

*end for*

*end for*

***End***

**SECTION III**

**METHODOLOGY**

This section elucidates the Block Floating-Point (BFP) representation of numerical values and expounds upon the BFP arithmetic used in our study. Additionally, it delineates the test images utilized therein.

**A. Introduction to Block floating point arithmetic:**

The representation of IEEE-754 standard 16-bit floating-point number is shown in Fig. 3.Detailed information about the floating point representation is explained in [//////reference].

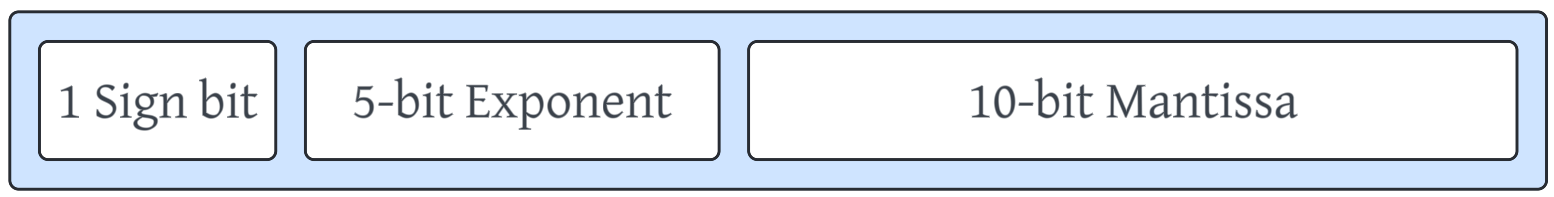


Fig.3 IEEE-754 16-bit Floating-point number representation

An N-data block represented with the BFP format consists of two parts: N mantissas and one exponent shared by the N numbers in a block. The process of the BFP conversion is defined as follows. Assuming that P is a data set containing N FP numbers, we can express the set as

P = (p1, …….., pi, ……., pN)

= (m1\*2e1, ……., mi\*2ei, ……, mN\*2eN ) (5)

The largest exponent in P is defined as the block exponent ep.

After deriving the common block exponent ep, the mantissa number mi is right shifter by di bits, where di = ep-ei. Thus, the BFP format of P, i.e., Pb is expressed as

Pb = (mb1, ….., mbi,…… mbN) \* 2ep (6)

where mbi = mi >> di is the BFP formatted mantissa.

**Implementation of BFP in our architecture:**

The research findings presented in Table I illustrate the hardware costs associated with various data precision operations [reference - 27]. The results indicate that fixed-point operations generally incur lower energy and area costs compared to floating-point (FP) operations, particularly noticeable in the case of addition operations.

/// [27] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," in Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC), Feb. 2014, pp. 10–14

////////

**TABLE 1: HARDWARE COSTS OF DIFFERENT DATA PRECISION OPERATIONS**

|  |  |  |
| --- | --- | --- |
| Operation | Energy (pJ) | Area(µm2) |
| 8b fixed Add | 0.03 | 36 |
| 32b fixed Add | 0.1 | 137 |
| 16b floating Add | 0.4 | 1360 |
| 32b floating Add | 0.9 | 4184 |
| 8b fixed Mult | 0.2 | 282 |
| 32b fixed Mult | 3.1 | 3495 |
| 16b floating Mult | 1.1 | 1640 |
| 32b floating Mult | 3.7 | 7700 |

Implementing a reduced bit-length data format holds substantial potential for optimizing the performance of CNN accelerators, particularly concerning chip area utilization, power efficiency, and memory requirements. By harnessing Binary Floating-Point (BFP), all multiply-accumulate operations within convolutional layers are executed in fixed-point format, thereby facilitating efficient processing while mitigating resource utilization.

In this paper, we are integrating Depth-wise convolution techniques with the strategic adoption of Binary Floating Point (BFP) format to enhance storage efficiency. For a set of n-numbers, each comprising a *Le* - bit exponent, a *Lm* -bit mantissa, and one sign bit, employing the BFP format yields an average bit length of *1 + Le/n + Lm* for storing the entirety of the data. This stands in contrast to the conventional floating-point representation, which necessitates *1 + Le + Lm* bits. Therefore, using BFP format for storing the Depth-wise kernels of dimensions *CI \* KW \* KH*, the average bit length per weight is *1+Le /(CI \* KW \* KH) + Lm.* Similarly, when storing the input feature maps, the average bit length per pixel is *1+Le /(CI \* KW \* KH \* OW \* OH) + Lm.* This formulation ensures that even for larger datasets, storage requirements are minimized while maintaining precision and accuracy.

For Depth-wise convolution, each channel of the input feature map forms a block and each kernel with respect to each output channel form a block. All numerical values in their respective block share a common block exponent but have their own mantissas. In this work, the bit length of block exponent is defined as 5, while the mantissa’s is 10.

**Validation of Methodology:**

The hardware of the proposed BFP arithmetic based convolution units is implemented and is operation is verified using its equivalent Python model. Several test images are considered from the University of the Southern California (USC)-SIPI Image database and kernels like edge detection kernel, Blur kernels and Laplacian kernels are used to verify the operation. The detailed report of the performance of the architecture is given in the section V.

**SECTION IV**

**HARDWARE IMPLEMENTATION**

In this paper, a reconfigurable architecture that supports a fixed kernel size of 3\*3 and an input image size of *IW \* IH (4 ≤ IW, 4 ≤ IH*) is proposed. Each pixel and weight are represented in 16-bit floating point format. Fig. 4 illustrates the top-level diagram of the architecture. Our architecture is composed of four preliminary components: a processing element, on-chip ping-pong buffers, external memory and a memory controller.

**A. Processing Element Architecture**

To initiate the convolution process, nine pixel values are fetched from on-chip buffers, coinciding with the nine weights engaged in the operation, are fed into the FP to BFP converters.

***FP to BFP converter:***

The FP to BFP converter, illustrated in Fig. 5, comprises three essential sub-blocks: the Max Exponent Generator, the Exponent Difference Generator, and the Shifter. Firstly, the Max Exponent Generator utilizes a successive comparison technique to determine the highest exponent among the input values. The Exponent Difference Generator calculates the difference between max exponent and the exponents of each input value, which is then utilized to right-shift all mantissa bits of each input value accordingly. FP to BFP converter produces nine 11-bit mantissa (MSB-Sign bit, 10 bits - aligned Mantissa) for each input value and 5-bit block exponent for both the pixels and weights.

***Multipliers:***

Nine signed binary multiplier units are employed to perform multiplication operations between the nine mantissas of the pixels and their corresponding weights. This process results in the generation of nine 21-bit products, each represented in sign-magnitude format.

***2’s Complement Adder:***

Following the multiplication, all the products are converted into 2’s complement form and added together. The final sum again converted back to sign magnitude form. Without any truncation, the sum of the block exponents of the 9 pixels and weights, along with the final sum in sign-magnitude form, constitutes the final value of the output node. This value is expressed in a 32-bit floating-point format and transferred to external memory for further processing.

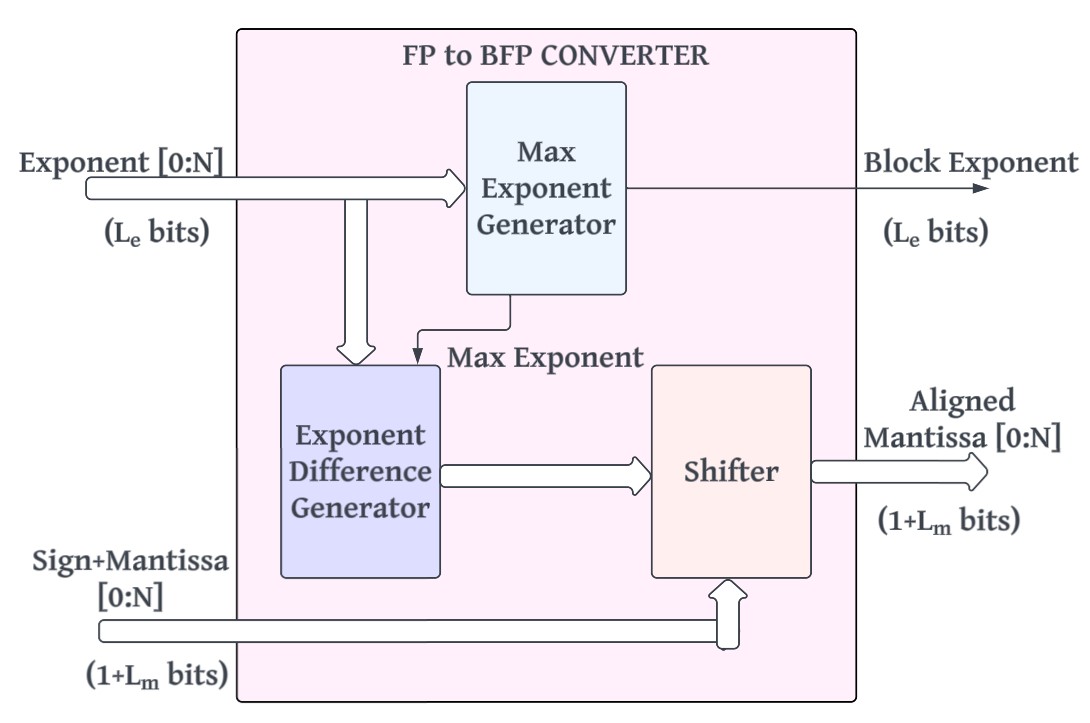


Fig. 5. FP to BFP Converter

**SECTION V**

**RESULTS**

**SECTION VI**

**CONCLUSION**