New Methods for Implementation of 2-D Convolution for Convolutional Neural Network (CNN)

Tokunbo Ogunfunmi
Signal Processing Research Lab (SPRL),
Electrical & Computer Engr. (ECEN) Dept.,
School of Engineering,
Santa Clara University.
September 2020

Outline

- Motivation
- Challenges in Implementing 2-D Convolution for CNNs
- Method #1
- Method #2
- Future Work
- Summary and Conclusions

Convolutional Neural Networks

- CNNs are most popular for vision tasks like image classification and segmentation.
- CNNs are computationally intensive.
- Computation and data movement requires energy.
  - Data read and write major energy consumer.
- Activations, partial sums and weights constitute the most amount of data moved.

2D Convolution Operation

- Weights multiplied by input feature map and accumulated.
- Kernel or weights are synonymous.
- Filters in CNNs convolve over multiple channels.

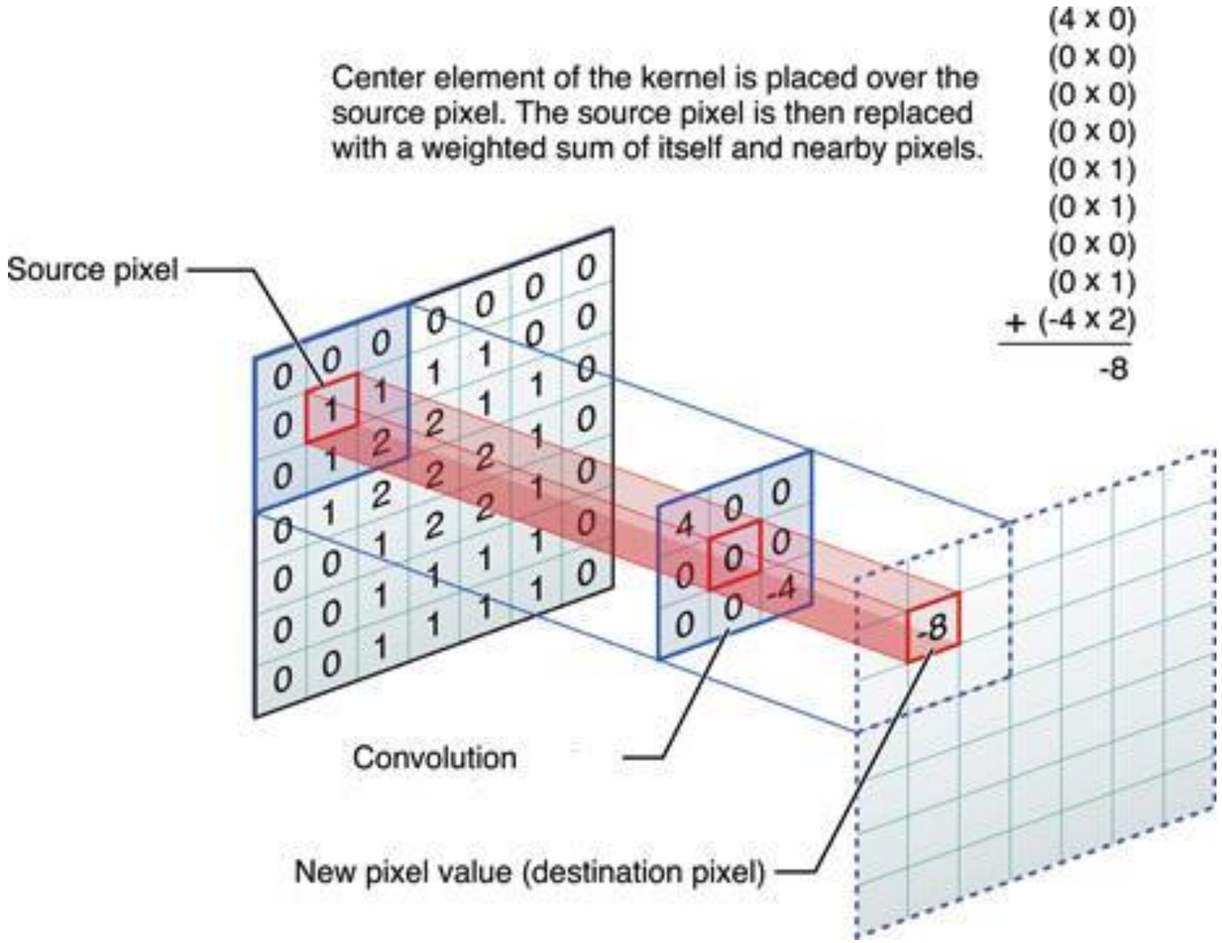
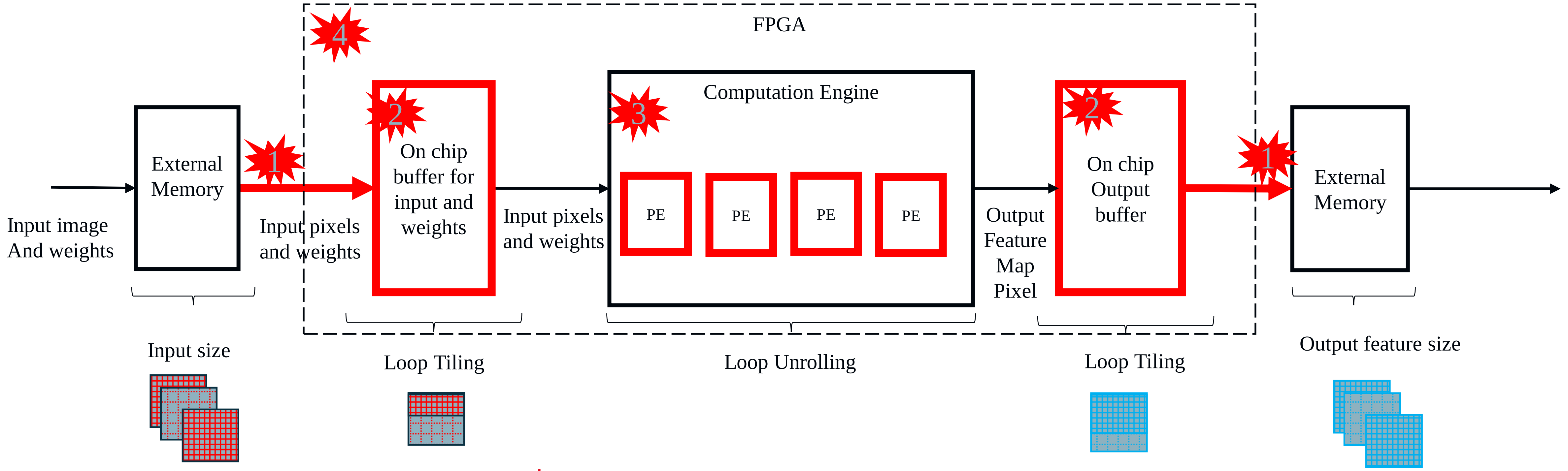


Image Source: https://cedar.buffalo.edu/~srihari/CSE574/Chap5/Chap5.5.6-ConvolutionalNetworks.pdf

Challenges in FPGA Implementation of DNNs



1 Challenge 1 : Huge Memory Transfer (Input and Output)

2 Challenge 2 : Large Onchip Buffers (Input and Output)

3 Challenge 3: Large Compute

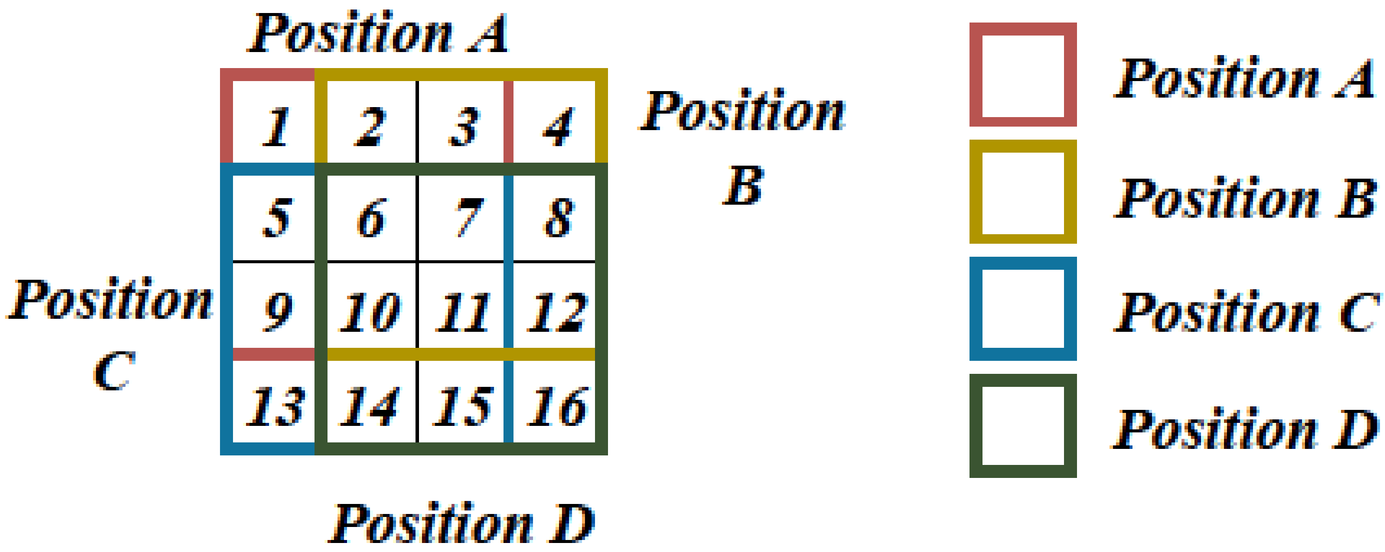
4 Challenge 4: Complicated Scheduling and Dataflow Control

Method #1
FIFO Based

Santa Clara University
School of Engineering

Convolution – Tile Based

- Conv. with 3x3 kernel
- Need to read at least 3 rows of pixels into line buffers.
- Better tile based processing with 4 line buffers.



Proposed Dataflow

- Method proposed for VGG16 which has only 3x3 kernel
  - Can be extended to other kernel sizes as well
- The proposed method aims to reduce the read and write bandwidth.
  - Aims to read the input feature map only once.

[4]. A. Ardakani, C. Condo, M. Ahmadi, and W. J. Gross, “An architecture to accelerate convolution in deep neural networks,” IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 4, pp. 1349–1362, 2017.

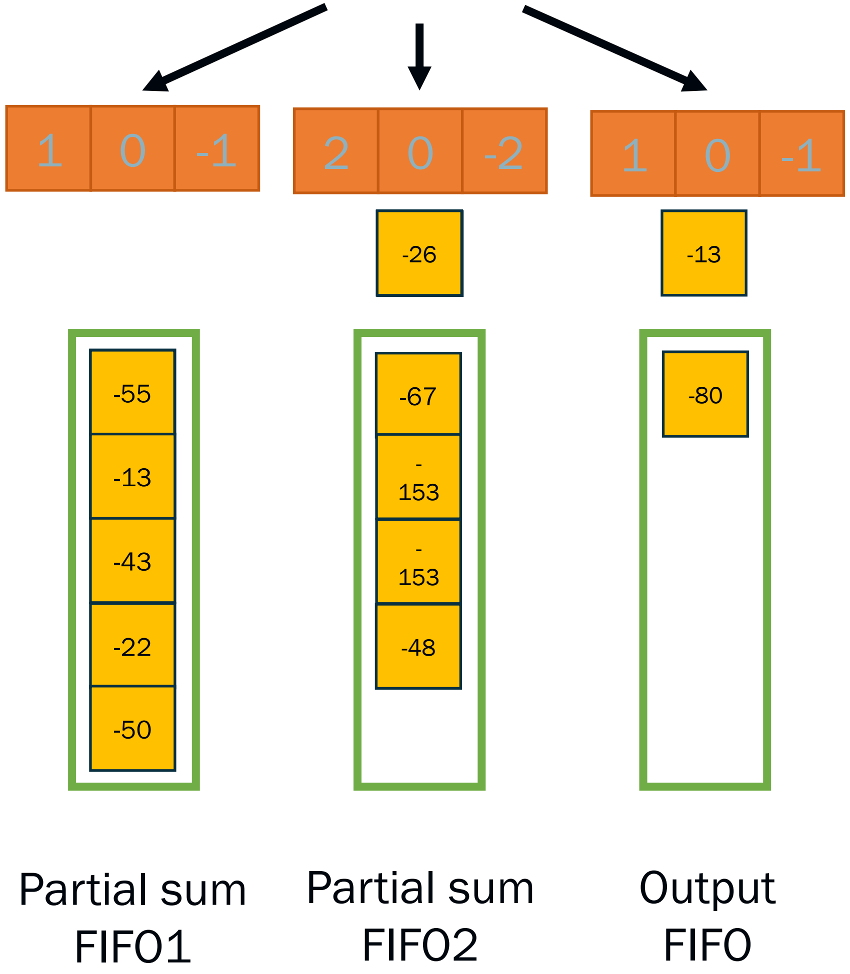
The Basic Idea and an Example

$$\begin{aligned} row_i, Conv_1 = {} & i_1 * w_1 + i_2 * w_2 + i_3 * w_3 + \\ & j_1 * w_4 + j_2 * w_5 + j_3 * w_6 + \\ & k_1 * w_7 + k_2 * w_8 + k_3 * w_9 \end{aligned} \tag{1}$$

$$row_i, psum1^1 = i_1 * w_1 + i_2 * w_2 + i_3 * w_3 \tag{2}$$

$$row_i, psum2^1 = j_1 * w_4 + j2 * w5 + j3 * w6 + row_i, psum1^1 \tag{3}$$

$$row_i, Conv_1 = k1 * w7 + k2 * w8 + k3 * w9 + row_i, psum2^1 \tag{4}$$



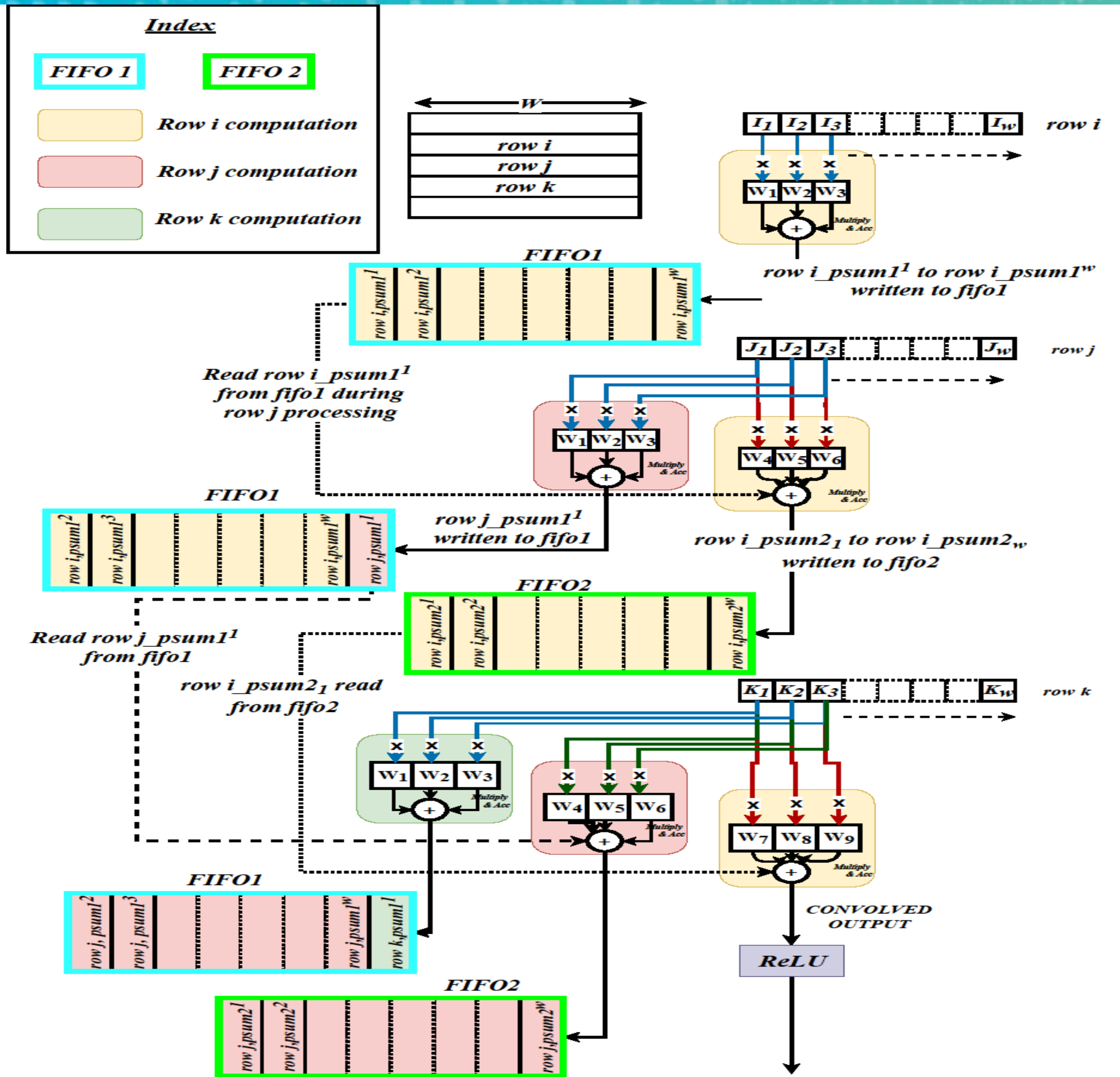
144	178	167	231	210
127	145	149	195	204
87	112	100	75	85
67	95	75	65	82
33	90	115	143	232

Santa Clara University
School of Engineering

Processing Element (PE)

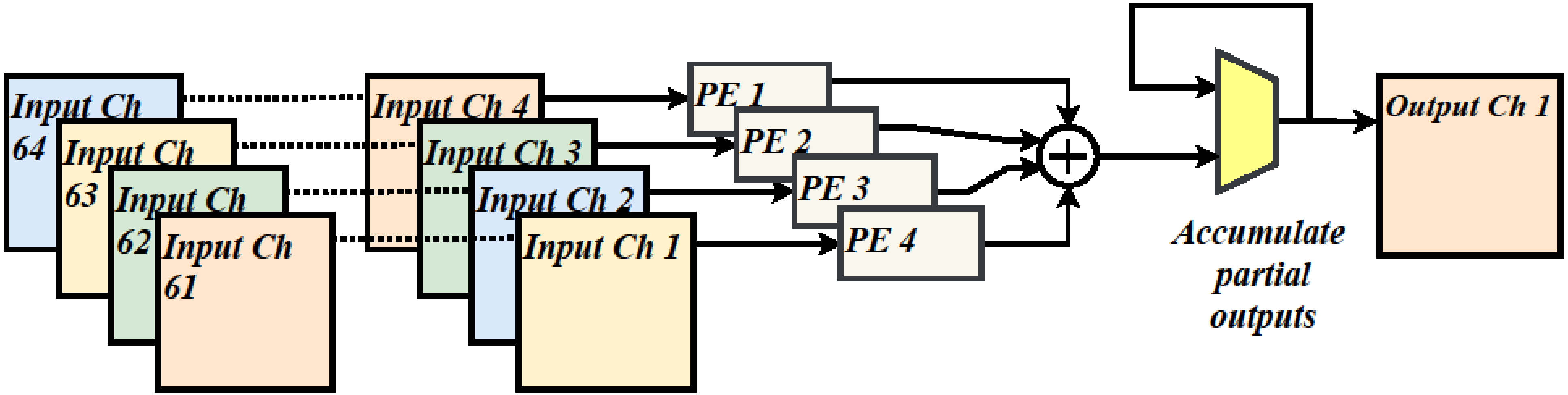
- Uses 3 FIFOs to compute convolution output
- Partial sums are stored in 2 FIFOs.
- 3rd FIFO used to accumulate outputs
- Partial sum FIFO size = width of the input image.
  - Rounded up to 256 in case of VGG16
  - 2 such FIFOs
- Output FIFO used to combine output of Processing elements
  - Output FIFO size for VGG16 =>256x256 = 64k

Processing Element (PE) Architecture



Parallel Implementation

- Example of how 64 channel Input Feature (IF) map is processed in groups of 4.



Hardware Platform

- XILINX PYNQZ1 has a ZNYQ 7000 soc.
- Has an ARM processor running at 650 MHz.
- Programable logic works at 100 MHz.
- Programmable logic can be controlled using Python code

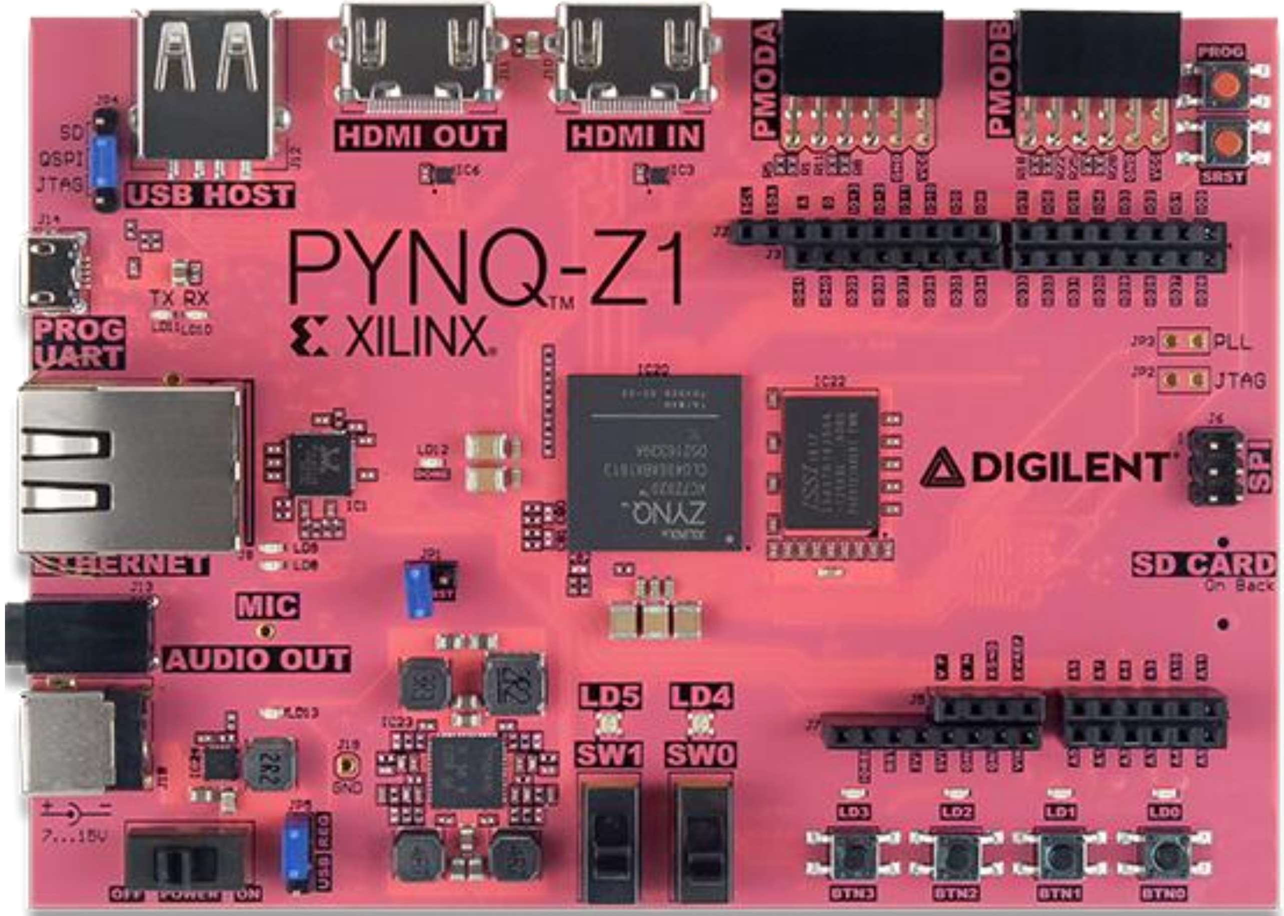


Image source : https://reference.digilentinc.com/_media/reference/programmable-logic/pynq-z1/pynq-z1-1.png

Santa Clara University
School of Engineering

Architecture Implementation

- The architecture was implemented using C++
- HLS used to convert C++ to hardware.
- RTL IP created and built into block design
- Xilinx Vivado used to synthesize, place and route the block design
- Fixed point 16 format was used for the weights and partial sums.
- Python code using PYNQ library used to implement the Conv. Layer operation.

FPGA Utilization

Includes the space required AXI DMAs, Block RAMs othe blocks.

Resource	Utilization	Available	Utilization%
LUT	19213	53200	36.11%
LUTRAM	2651	17400	15.24%
FF	25428	106400	23.90%
BRAM	93.50	140	66.79%
DSP	75	220	34.09%

Table 2. Utilization Summary of the implementation on ZYNQ XC7Z020-1CLG400C

Results and Comparisons

	JSSC'17 [10]	Ardakani et al[4]	This work	
Nominal Frequency(Mhz)	200	400	100	
Layer	Total Latency(ms)	Total Latency(ms)	Total Latency(ms)	Conv. Processing Latency(ms)
Conv1-1	76.2	72.9	118.4	32.1
Conv1-2	910.3	1555.2	321.4	256.9
Conv2-1	470.3	784.5	246.8	128.4
Conv2-2	894.3	1564.9	373.2	256.9
Conv3-1	241.1	798.2	348.3	128.4
Conv3-2	460.9	1596.5	479.1	256.9
Conv3-3	457.7	1596.	475.7	256.9
Conv4-1	135.8	825.7	597.1	128.4
Conv4-2	254.8	1651.5	691.1	256.9
Conv4-3	246.3	1651.	692.5	256.9
Conv5-1	54.3	440.4	594.3	64.2
Conv5-2	53.7	440.4	595.1	64.2
Conv5-3	53.7	440.4	595.2	64.2
Total VGG16	4309.5	13422.6	6128.8	2151.5

Table 1. Performance comparison for VGG16 benchmark

[10] Y. Chen, T. Krishna, J.S. Emer and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks" IEEE Journal of Solid State Circuits, vol. 52, no. 1, pp. 127-138, January 2017.

[4]. A. Ardakani, C. Condo, M. Ahmadi, and W. J. Gross, "An architecture to accelerate convolution in deep neural networks," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 4, pp. 1349–1362, 2017.

Method #2
Single Partial Product 2-D
(SPP2D)

Santa Clara University
School of Engineering

Convolution Operation

i0*w0	i1*w0	i2*w0	i3*w1	i4*w2
i5*w0	i6*w0	i7*w0	i8*w1	i9*w2
i10*w0	i11*w0	i12*w0	i13*w1	i14*w2
i15*w3	i16*w3	i17*w3	i18*w4	i19*w5
i20*w6	i21*w6	i22*w6	i23*w7	i24*w8

Input

w0	w1	w2
w3	w4	w5
w6	w7	w8

Kernel

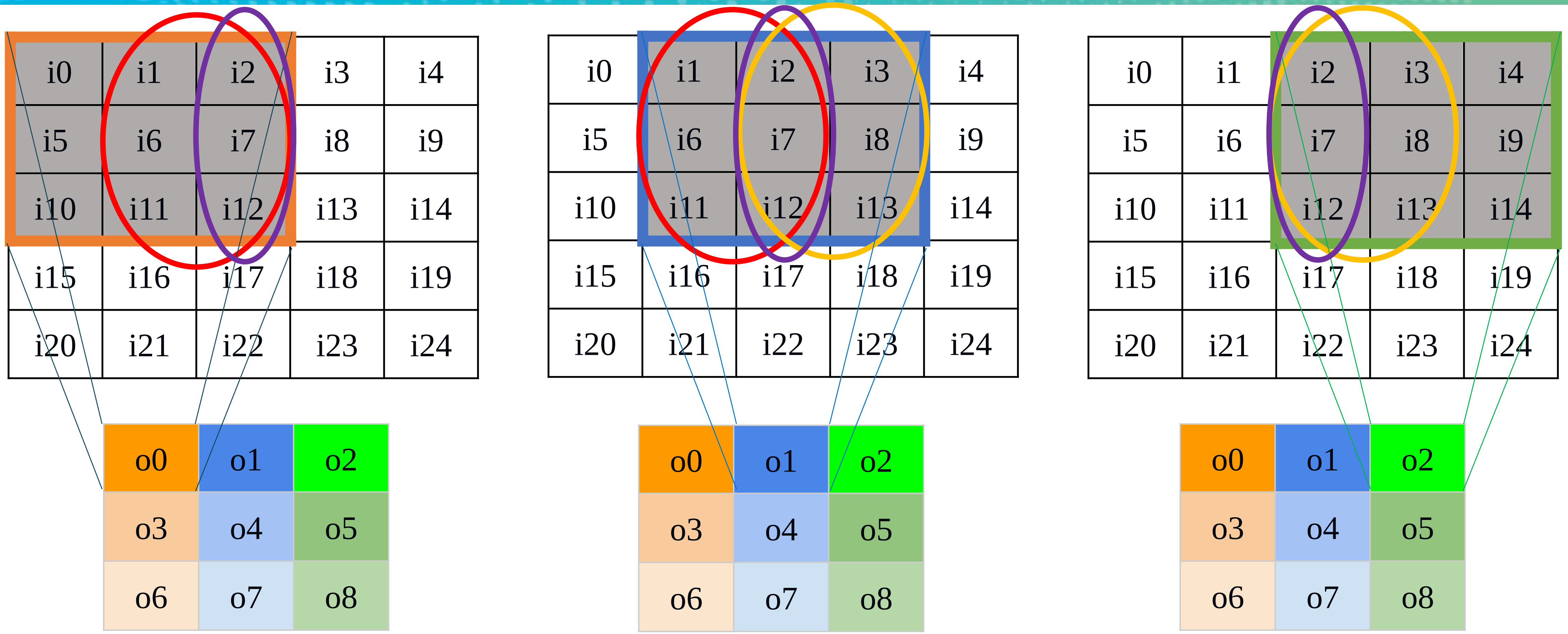
Σ

o0	o1	o2
o3	o4	o5
o6	o7	o8

Output

Consider and input of size 5x5,kernel of size 3x3.We consider a convolution operation with stride 1 and with zero padding.

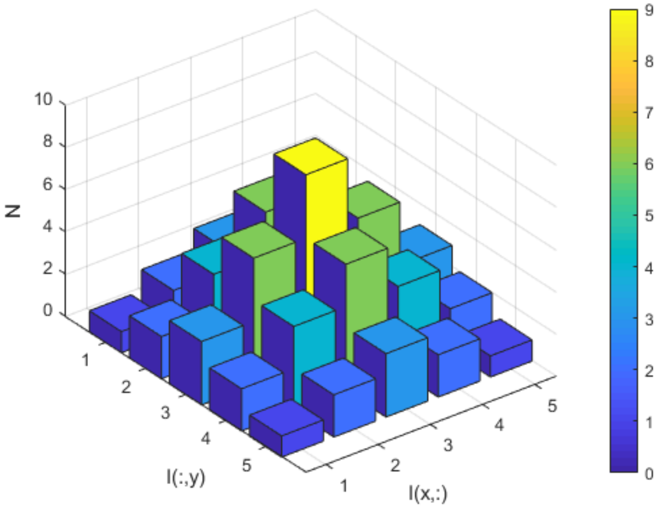
Convolution Operation



Frequency of use of an input pixels is N(x) where x is the frequency itself. For example i0 has frequency 1

Convolution Operation

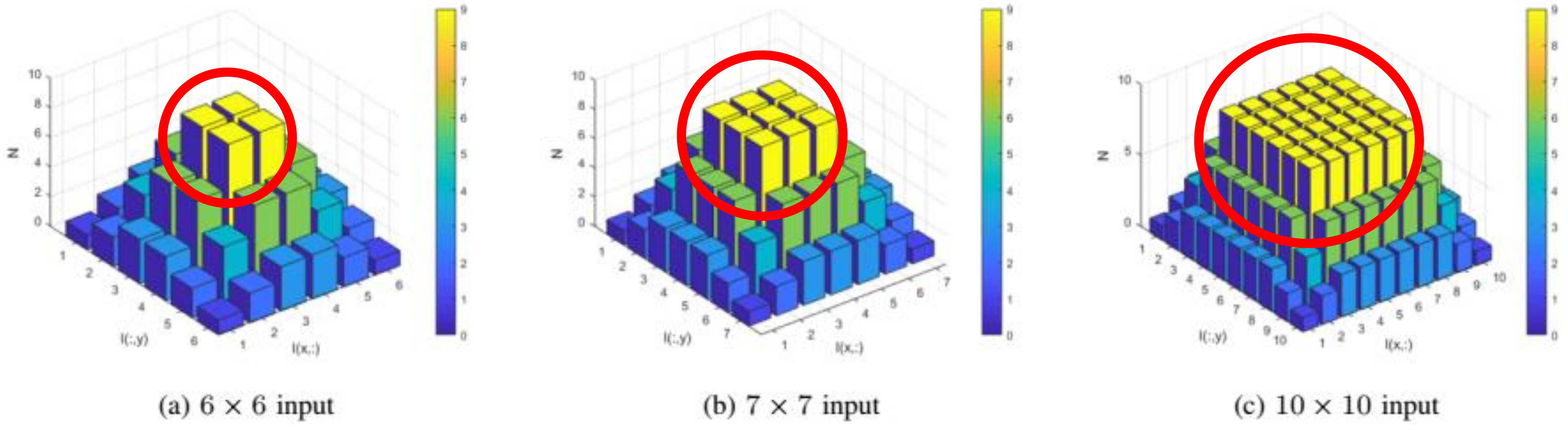
i0 N(1)	i1 N(2)	i2 N(3)	i3 N(2)	i4 N(1)
i5 N(2)	i6 N(4)	i7 N(6)	i8 N(4)	i9 N(2)
i10 N(3)	i11 N(6)	i12 N(9)	i13 N(6)	i14 N(3)
i15 N(2)	i16 N(4)	i17 N(6)	i18 N(4)	i19 N(2)
i20 N(1)	i21 N(2)	i22 N(3)	i23 N(2)	i24 N(1)



We use the notation N(x) to convey the frequency of use for an input pixel, here x is the frequency.

For example, pixel i12 has frequency N(9). It is the input pixel that is used 9 times with all 9 kernel elements.

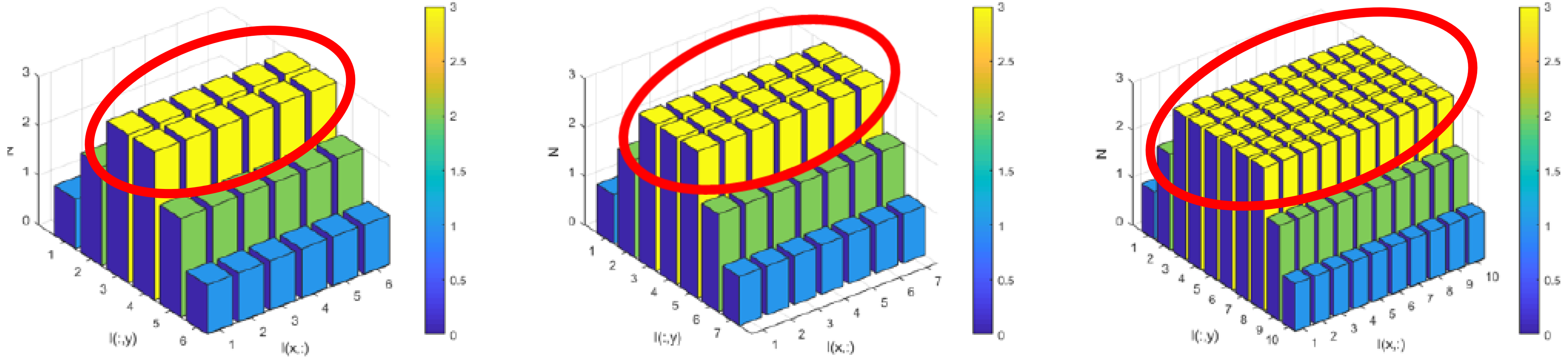
Pattern of Input Pixel Frequency in Sliding Window



Pattern of the frequency with which input pixels are needed in the existing* implementation

N(9) pixels always lies in the center of the input ((N-4)x(N-4) where N is input dimension) while all the other frequencies lie on the periphery boundary which is two pixels deep.

Patterns in Existing Implementation



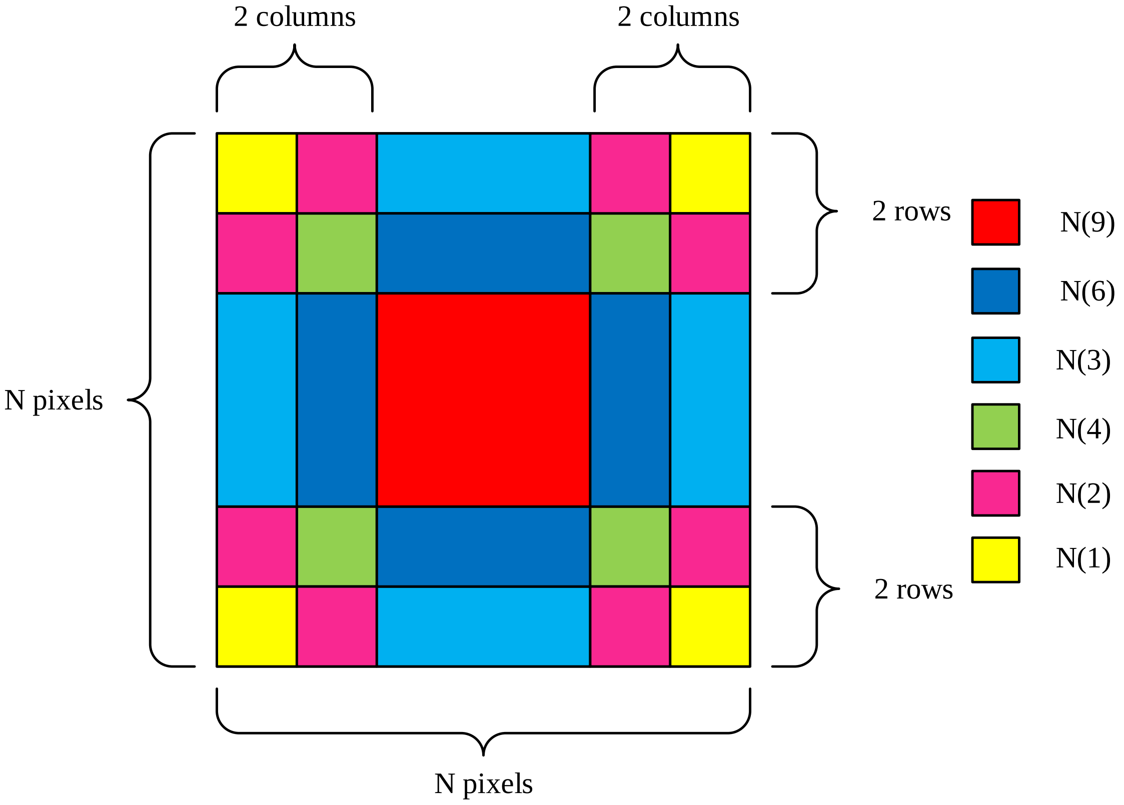
(a) 6×6 input (b) 7×7 input (c) 10×10 input

Pattern of the frequency with which input pixels are needed in the existing* implementation

N(3) pixels always lies in the center of the input while all the other frequencies lie on top and bottom and are two pixels deep

A. Ardakani, C. Condo, M. Ahmadi, and W. J. Gross, “An architecture to accelerate convolution in deep neural networks,” IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 4, pp. 1349–1362, 2017.

Generalized Equation for Pattern of Input Pixels for Sliding Window Operation



H_{input} and W_{input} are dimension of input and are N pixels in this example

Number of inputs with N(x)	Generalized Expression
N(9)	$(H_{input} – 2) \times (W_{input} – 2)$
N(6) and N(3)	$((H_{input} – 4) \times 2) + ((W_{input} – 4) \times 2)$
N(4) and N(1)	4
N(2)	2x4

input size	N(9)	N(6) and N(3)	N(4) and N(1)	N(2)
5	1	4	4	8
6	4	8	4	8
7	9	12	4	8
10	36	24	4	8
14	100	40	4	8
28	576	96	4	8
56	2704	208	4	8
112	11664	432	4	8
224	48400	880	4	8

SPP2D – Input stream

clock cycles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	N(9)	N(6)	N(6)	N(6)	N(6)	N(3)	N3)	N(3)	N(3)	N(4)	N(4)	N(4)	N(4)	N(2)	N(2)	N(2)	N(2)	N(2)	N(2)	N(2)	N(2)	N(1)	N(1)	N(1)	N(1)
	i12	i7	i11	i17	i13	i2	i14	i22	i10	i6	i8	i18	i16	i1	i13	i5	i9	i15	i19	i21	i23	i0	i4	i20	i24
w0	w0i12	w0i7	w0i11			w0i2			w0i10	w0i6				w0i1		w0i5						w0i0			
w1	w1i12	w1i7	w1i11		w1i13	w1i2				w1i6	w1i8			w1i1	w1i3										
w2	w2i12	w2i7			w2i13	w2i2	w2i14				w2i8				w2i3		w2i9						w2i4		
w3	w3i12	w3i7	w3ii11	w3i17					w3i10	w3i6			w3i16			w3i5		w3i15							
w4	w4i12	w4i7	w4i11	w4i17	w4i13					w4i6	w4i8	w4i18	w4i16												
w5	w5i12	w5i7		w5i17	w5i13		w5i14				w5i8	w5i18					w5i9		w5i19						
w6	w6i12		w6i11	w6i17				w6i22	w6i10				w6i16					w6i15		w6i21				w6i20	
w7	w7i12		w7i11	w7i17	w7i13			w7i22				w7i18	w7i16							w7i21	w7i23				
w8	w8i12			w8i17	w8i13		w8i14	w8i22				w8i18							w8i19		w8i23				w8i24

- Only i12 occupies all the multipliers with the 9 weight
- Complementary Sets : (i7,i22), (i17,i2), (i11,i 14), (i6,i19,i21,i24), (i16,i1,i19,i4), (i13, i10), (i8,i5,i23,i20), (i18,i3,i15,i0)

SPP2D – Optimized Input stream

	clock cycles	1	2	3	4	5	6	7	8	9
	N(x)	N(4),N(2),N(1)	N(4),N(2),N(1)	N(6),N(3)	N(4),N(2),N(1)	N(4),N(2),N(1)	N(6),N(3)	N(6),N(3)	N(6),N(3)	N(9)
weights	Complementary sets	i18+i3+i15+i0	i16+i1+i19+i4	i17 +i2	i8+i5 +i23+i20	i6+i19+i21+i24	i7 +i22	i13 + i10	i11 +i14	i12
w0		w0i0	w0i1	w0i2	w0i5	w0i6	w0i7	w0i10	w0i11	w0i12
w1		w1i3	w1i1	w1i2	w1i8	w1i6	w1i7	w1i13	w1i11	w1i12
w2		w2i3	w2i4	w2i2	w2i8	w2i9	w2i7	w2i13	w2i14	w2i12
w3		w3i15	w3i16	w3i17	w3i5	w3i6	w3i7	w3i10	w3ii11	w3i12
w4		w4i18	w4i16	w4i17	w4i8	w4i6	w4i7	w4i13	w4i11	w4i12
w5		w5i18	w5i19	w5i17	w5i8	w5i9	w5i7	w5i13	w5i14	w5i12
w6		w6i15	w6i16	w6i17	w6i20	w6i21	w6i22	w6i10	w6i11	w6i12
w7		w7i18	w7i16	w7i17	w7i23	w7i21	w7i22	w7i13	w7i11	w7i12
w8		w8i18	w8i19	w8i17	w8i23	w8i24	w8i22	w8i13	w8i14	w8i12

Two benefits of combining input pixels into complementary sets

1. All multipliers are occupied
2. Arrive at output faster. Theoretically in 9 cycles for this arrangement

i0	i1	i2	i3	i4
i5	i6	i7	i8	i9
i10	i11	i12	i13	i14
i15	i16	i17	i18	i19
i20	i21	i22	i23	i24

Input

w0	w1	w2
w3	w4	w5
w6	w7	w8

Kernel

o0	o1	o2
o3	o4	o5
o6	o7	o8

Output

SPP2D – Partial Products Sorted into their Outputs

i18+i3+i15+i0	6	4	4	2	3	3	2	3	3
i16+i1+i19+i4	5	5	6	1	1	3	1	1	3
i17 +i2	7	7	7	8	8	8	8	8	8
i8+i5 +i23+i20	8	1	1	8	1	1	9	2	2
i6+i19+i21+i24	2	2	5	2	2	5	9	9	4
i7 +i22	8	8	8	8	8	8	10	10	10
i13 + i10	5	9	9	5	9	9	5	9	9
i11 +i 14	2	2	8	2	2	8	2	2	8
i12	3	3	3	3	3	3	3	3	3

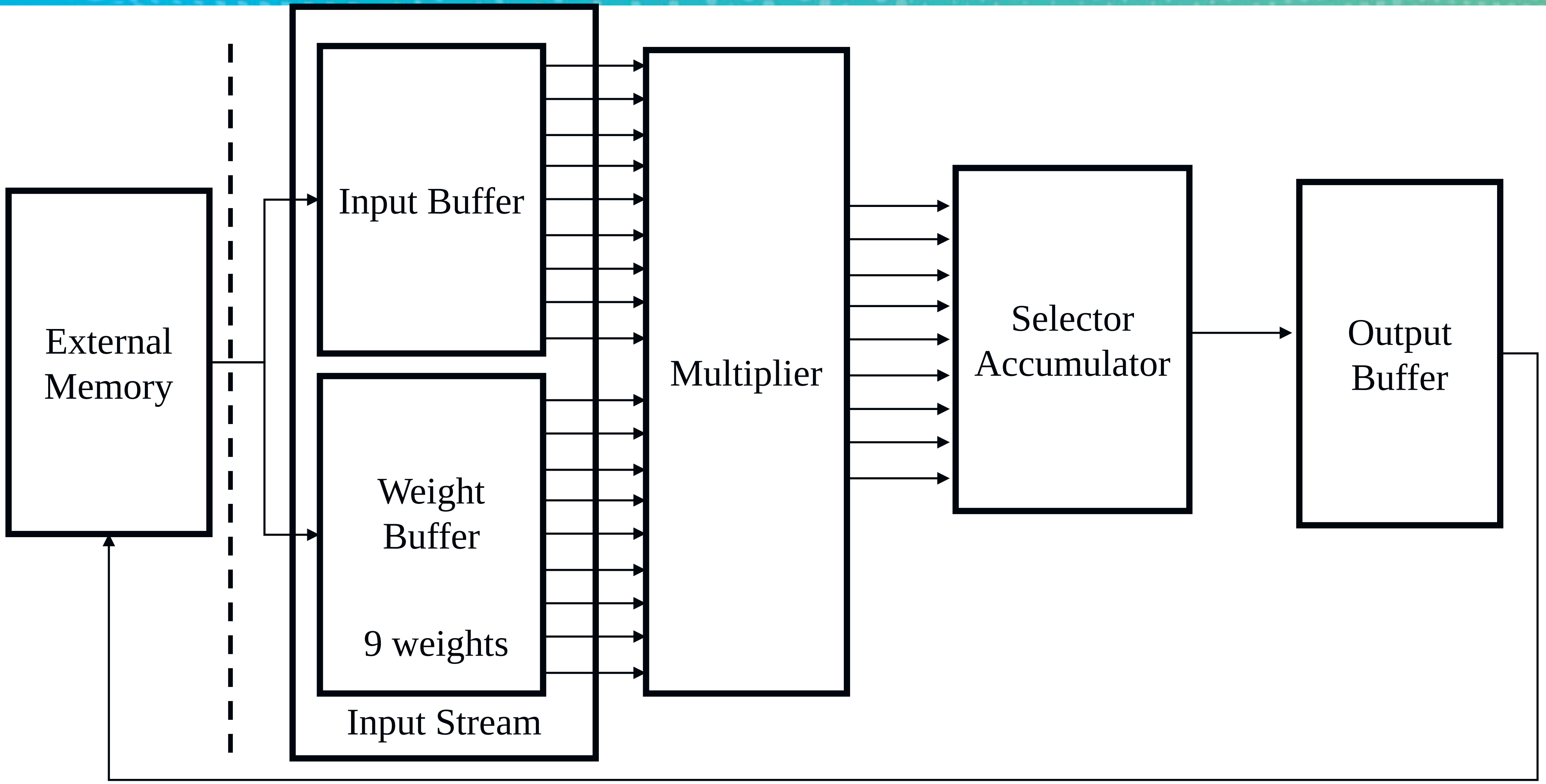
	w0	w1	w2	w3	w4	w5	w6	w7	w8
	2	**2**	**3**	**1**	**2**	**1**	**1**	**3**	**1**
Σ	**12**	**10**	**21**	**8**	**4**	**8**	**5**	**6**	**3**

o0 **77**	o1 **75**	o2 **93**
o3 **69**	o4 **68**	o5 **82**
o6 **81**	o7 **98**	o8 **85**

Output

The highlighted partial products in red contribute to the first output pixel

SPP2D – Hardware Architecture



SPP2D – Hardware Architecture

- Delivers output in 9 cycles for an input of 5x5 and kernel of size 3x3.
- Architecture involves blowing up an input matrix of 25 pixels to 81 pixels.
- The selector accumulator for this example is designed for a 5x5 input and 3x3 weights. Need to scale it to an input size of 224x224 for VGG16 example.

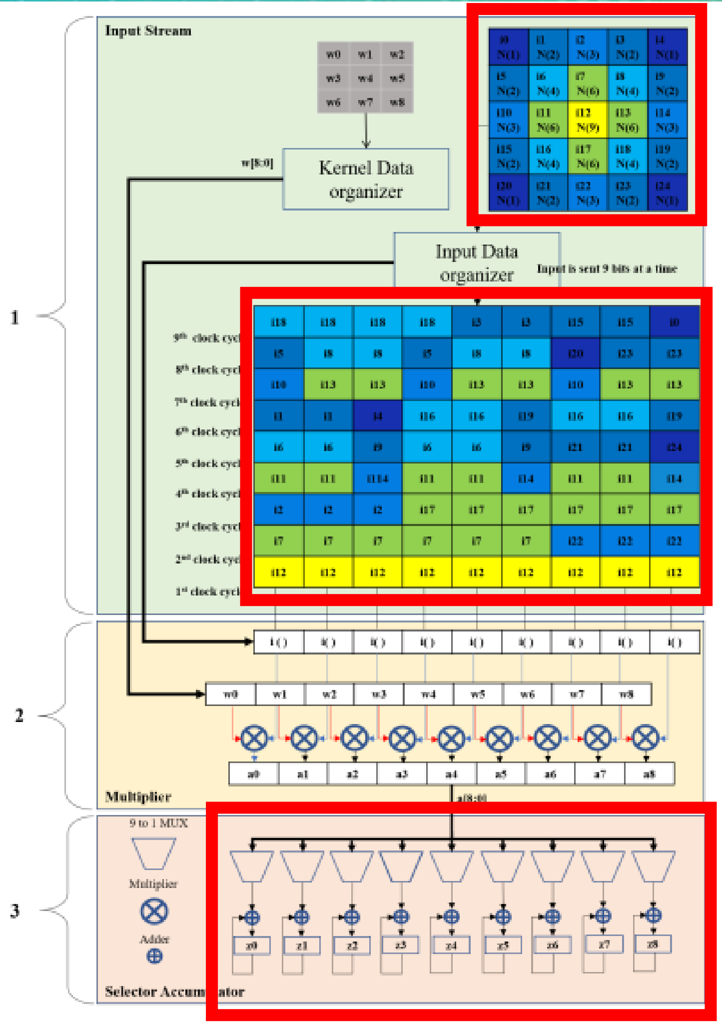


Fig. 4: Architecture to perform SPP2D Convolution

5x5 input results
25 pixels

Would require a big buffer to accommodate 81 pixels

The mux selector accumulator needs to scale to an input of size 224x224

Results and Comparisons

	without padding							with padding						
	a	b	c	a-c	a/c	b-c	b/c	d	e	f	d-f	d/f	e-f	e/f
input size	Sliding Window (w/o padding)	[1] (w/o padding)	SPP2D Conv w/o padding					Sliding Window (w padding)	[1] (w padding)	SPP2D Conv (w padding)				
5	81	45	9	72	9.00	36	5.00	225	105	25	200	9.00	80	4.20
6	144	72	16	128	9.00	56	4.50	324	144	36	288	9.00	108	4.00
7	225	105	25	200	9.00	80	4.20	441	189	49	392	9.00	140	3.86
10	576	240	64	512	9.00	176	3.75	900	360	100	800	9.00	260	3.60
14	1296	504	144	1152	9.00	360	3.50	1764	672	196	1568	9.00	476	3.43
28	6084	2184	676	5408	9.00	1508	3.23	7056	2520	784	6272	9.00	1736	3.21
56	26244	9072	2916	23328	9.00	6156	3.11	28224	9744	3136	25088	9.00	6608	3.11
112	108900	36960	12100	96800	9.00	24860	3.05	112896	38304	12544	100352	9.00	25760	3.05
224	443556	149184	49284	394272	9.00	99900	3.03	451584	151872	50176	401408	9.00	101696	3.03

Our Algorithm is 9x faster than the sliding window and 3x faster than the Warren Gross Implementation

[1] A. Ardakani, C. Condo, M. Ahmadi, and W. J. Gross, “An architecture to accelerate convolution in deep neural networks,” IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 4, pp. 1349–1362, 2017.

Future Work

Santa Clara University
School of Engineering

Future work (1)

- Use Compression: CNNs can be compressed to INT8 with minimal impact on accuracy.
  - More Processing Elements (PEs) can be implemented.
  - Faster operation
- Compress weights and activations to reduce bandwidth requirement.
- The utilization percentage of Method #1 FIFOs for the later layers of the CNNs is low

Future work (2)

- Better utilization of FIFOs for later layers of CNNs of Method #1.
- Better utilization of Multipliers for layers of CNNs of Method #2.
- These two methods can be utilized for other non-FPGA platforms e.g. ASICs, CPUs, GPUs, etc.
- Demonstrate scalability to practical sizes such as 224x224.

Summary and Conclusions

Summary and Conclusions

- We presented **two new methods** for 2-D convolution that offer considerable reduction in power, computational complexity and efficiency offering a considerably better architecture.
- The **first method** is based on using FIFOs and computes convolution results using row-wise inputs as opposed to traditional tile-based processing giving considerably reduced latency.
- The **second method** Single Partial Product 2-D (SPP2D) Convolution prevents recalculation of partial weights and reduces input reuse.
- Hardware implementation results with improvements are presented.

Santa Clara University
School of Engineering

References & Acknowledgements

Reference 1

[A FIFO Based Accelerator for CNNs]

Reference 2

[A Fast 2-D Convolution Technique for Deep Neural Networks]

Acknowledgements

Xilinx University Program

Vineet Panchbaiyye, Santa Clara University

Anaam Ansari, Santa Clara University

Santa Clara University
School of Engineering

Questions & Answers

Contact Information:

Tokunbo Ogunfunmi
Santa Clara University
Email: Togunfunmi@scu.edu