A Fast 2-D Convolution Technique for Deep Neural Networks

Anaam Ansari, Prof. Tokunbo Ogunfunmi

Signal Processing Research Lab. (SPRL),

Department of Electrical Engineering,

Santa Clara University, CA 95053, USA

2020 IEEE International Symposium on Circuits and Systems
Virtual, October 10-21, 2020

Outline

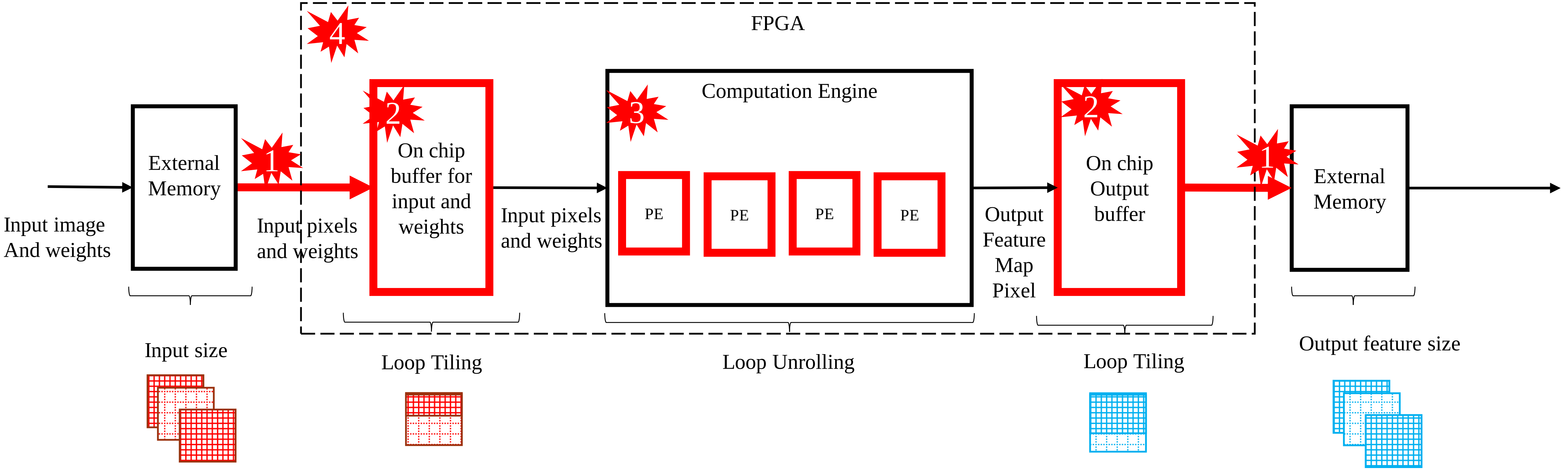
- Part 1
  - Motivation
  - Convolution operation
- Part 2
  - Patterns in Sliding Window
  - Patterns in Existing Implementation*
  - Single Partial Product 2-D convolution
- Part 3
  - Architecture proposed in ISCAS 2020
  - Results
- Conclusion

*Ardakani, Arash, et al. "An architecture to accelerate convolution in deep neural networks." IEEE Transactions on Circuits and Systems I: Regular Papers 65.4 (2017): 1349-1362.

Part 1

- Motivation
- Convolution operation

Motivation



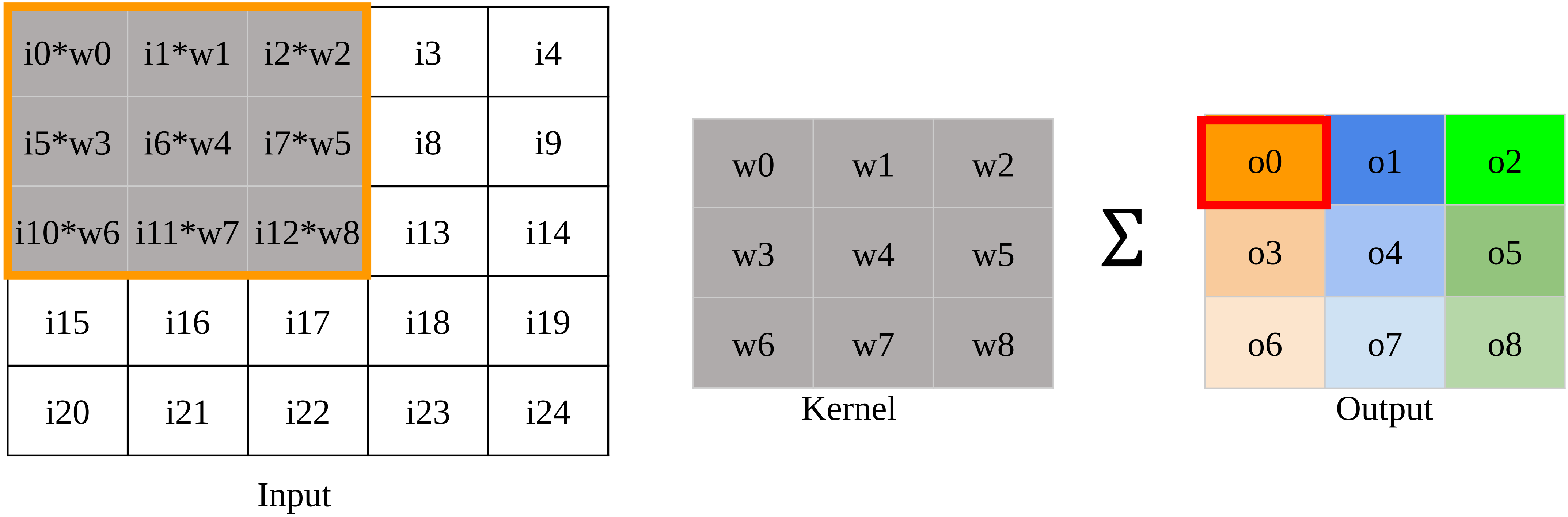
1 Challenge 1 : Huge Memory Transfer (Input and Output)

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

3 Challenge 3: Large Compute

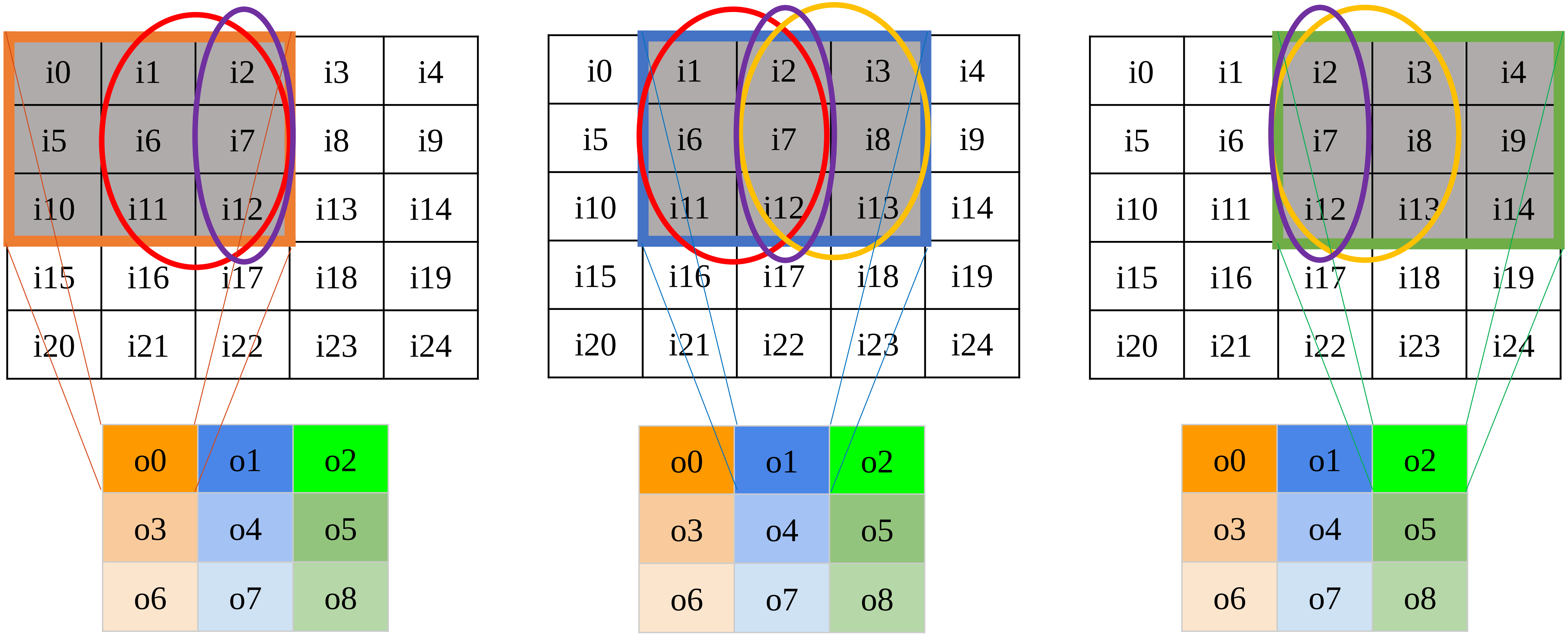
4 Challenge 4: Complicated Scheduling and Dataflow Control

Convolution Operation



Consider and input of size 5x5,kernel of size 3x3.We consider a convolution operation with stride 1 and with no padding.

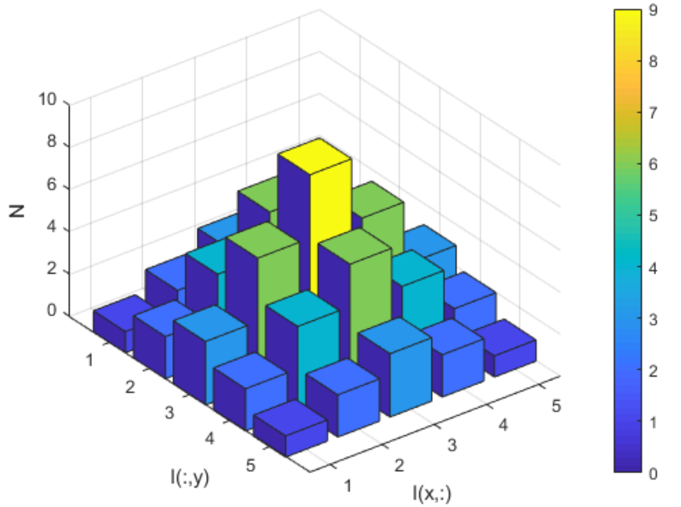
Convolution Operation



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)

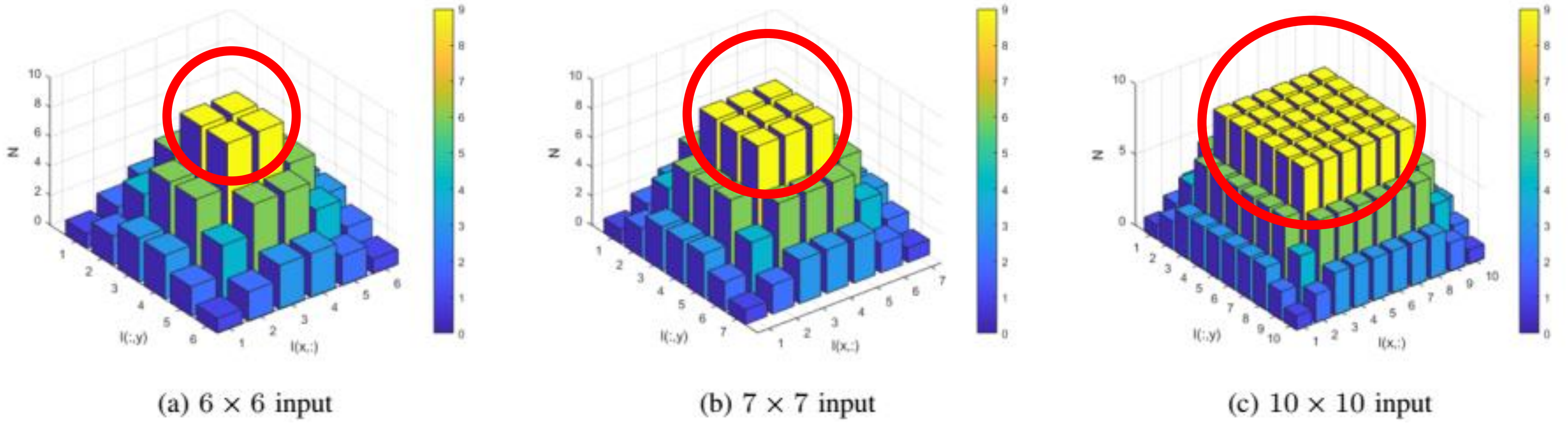


We use the notation N(x) to convey the frequency of use for an input pixel, here x is the frequency. For example, pixel i12 has frequency N(9). It is the input pixel that is used 9 times with all 9 kernel elements.

Part 2

- Patterns in Sliding Window
- Patterns in Existing Implementation*
- Single Partial Product 2-D convolution

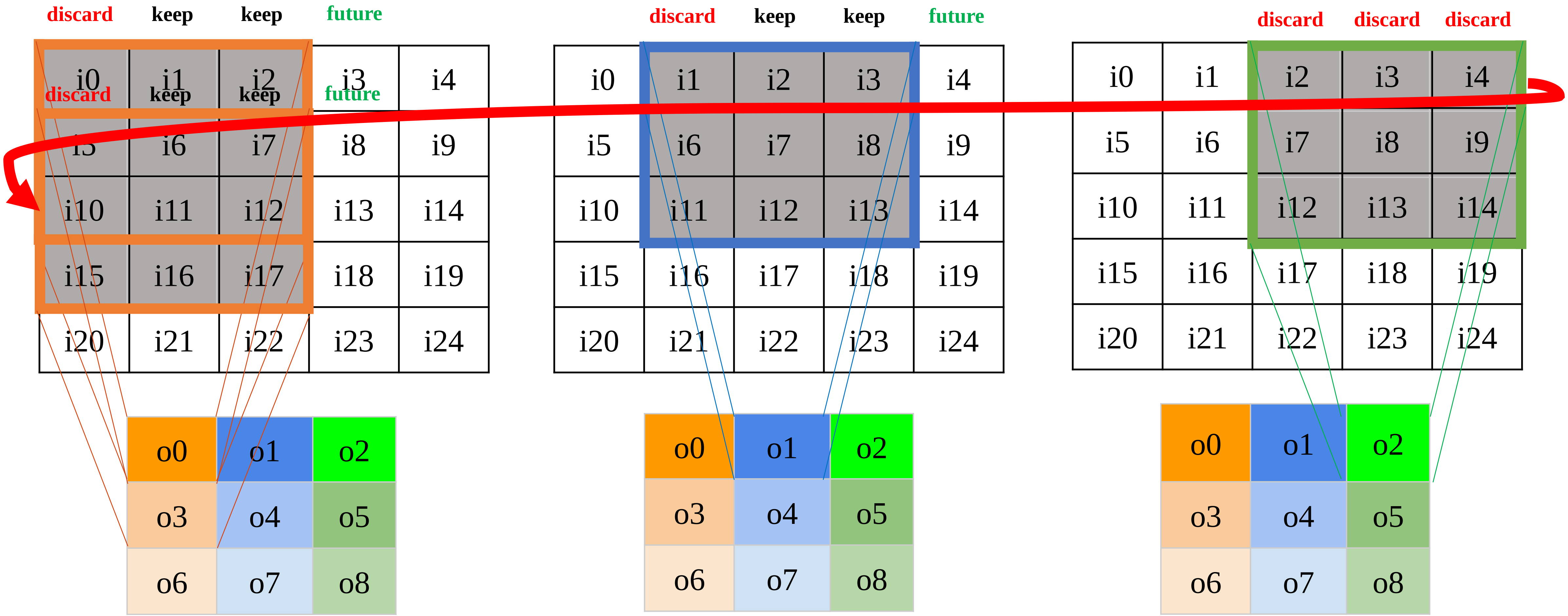
Pattern in Sliding Window



Pattern of the frequency with which input pixels are needed in a sliding window operation

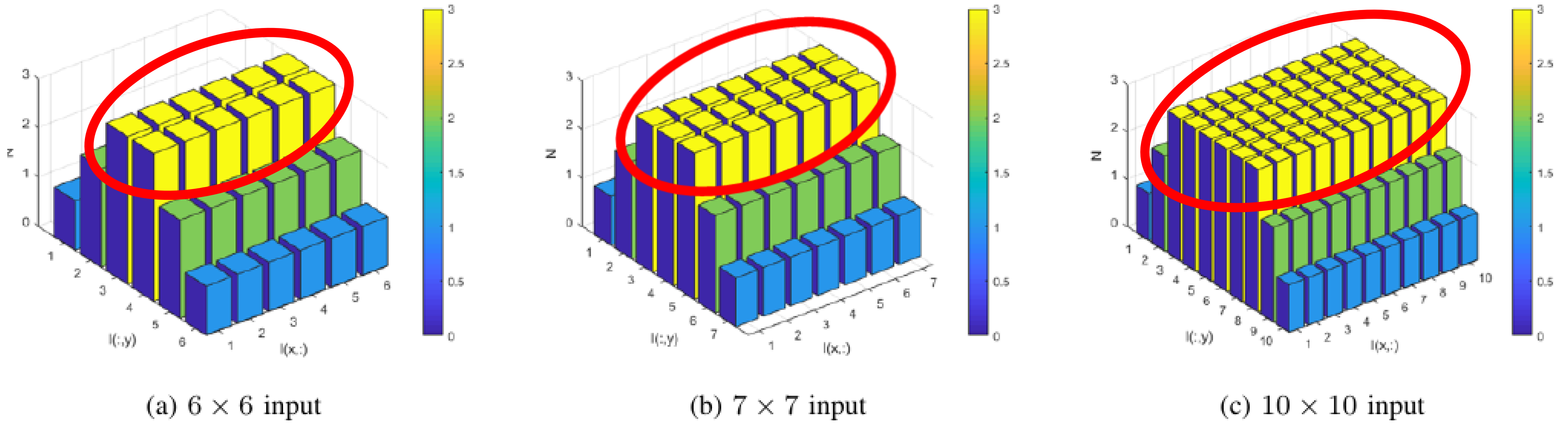
N(9) pixels always lies in the center of the input ((N-4)x(N-4) where N is input dimension) while all the other frequencies lie on the periphery boundary which is two pixels deep.

Existing Implementation



*Ardakani, Arash, et al. "An architecture to accelerate convolution in deep neural networks." IEEE Transactions on Circuits and Systems I: Regular Papers 65.4 (2017): 1349-1362.

Patterns in Existing Implementation

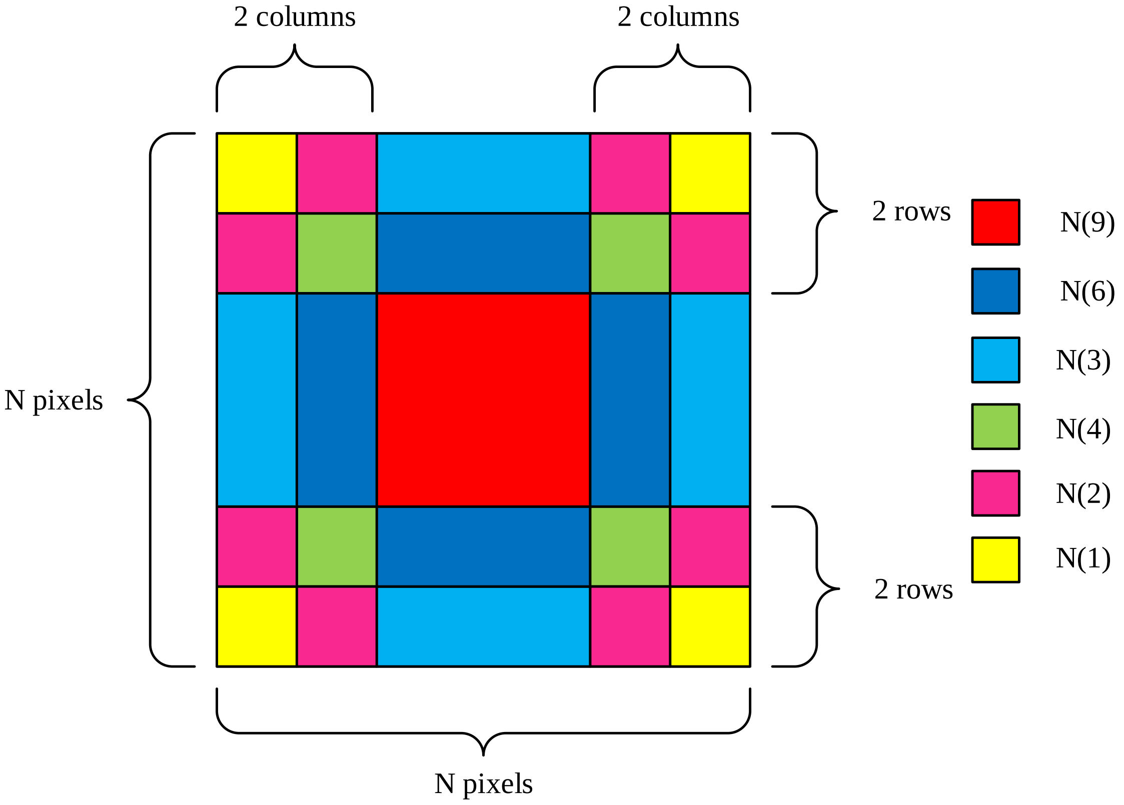


Pattern of the frequency with which input pixels are needed in the existing* implementation

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



N(x)	Generalized Expression for no of pixels with N(x)
N(9)	$(H_{input} – 2) \times (W_{input} – 2)$
N(6) and N(3)	$((H_{input} – 4) \times 2) + ((W_{input} – 4)\times2)$
N(4) and N(1)	4
N(2)	2x4

	Number of input pixels corresponding to N(x)			
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

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

SPP2D – Input stream

clock cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
	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)
	i12	i7	i11	i17	i13	i2	i14	i22	i10	i6	i8	i18	i16
w0	w0i12	w0i7	w0i11			w0i2			w0i10	w0i6			
w1	w1i12	w1i7	w1i11		w1i13	w1i2				w1i6	w1i8		
w2	w2i12	w2i7			w2i13	w2i2	w2i14				w2i8		
w3	w3i12	w3i7	w3ii11	w3i17					w3i10	w3i6			w3i16
w4	w4i12	w4i7	w4i11	w4i17	w4i13					w4i6	w4i8	w4i18	w4i16
w5	w5i12	w5i7		w5i17	w5i13		w5i14				w5i8	w5i18	
w6	w6i12		w6i11	w6i17				w6i22	w6i10				w6i16
w7	w7i12		w7i11	w7i17	w7i13			w7i22				w7i18	w7i16
w8	w8i12			w8i17	w8i13		w8i14	w8i22				w8i18	

clock cycles	14	15	16	17	18	19	20	21	22	23	24	25
	N(2)	N(2)	N(2)	N(2)	N(2)	N(2)	N(2)	N(2)	N(1)	N(1)	N(1)	N(1)
	i1	i3	i5	i9	i15	i19	i21	i23	i0	i4	i20	i24
w0	w0i1		w0i5						w0i0			
w1	w1i1	w1i3										
w2		w2i3		w2i9						w2i4		
w3			w3i5		w3i15							
w4												
w5				w5i9		w5i19						
w6					w6i15		w6i21				w6i20	
w7							w7i21	w7i23				
w8						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 Convolution: Input, Kernel, Output

i0 6	i1 5	i2 7	i3 4	i4 6
i5 8	i6 2	i7 8	i8 1	i9 5
i10 5	i11 2	i12 3	i13 9	i14 8
i15 2	i16 1	i17 8	i18 3	i19 3
i20 9	i21 9	i22 10	i23 2	i24 4

Input

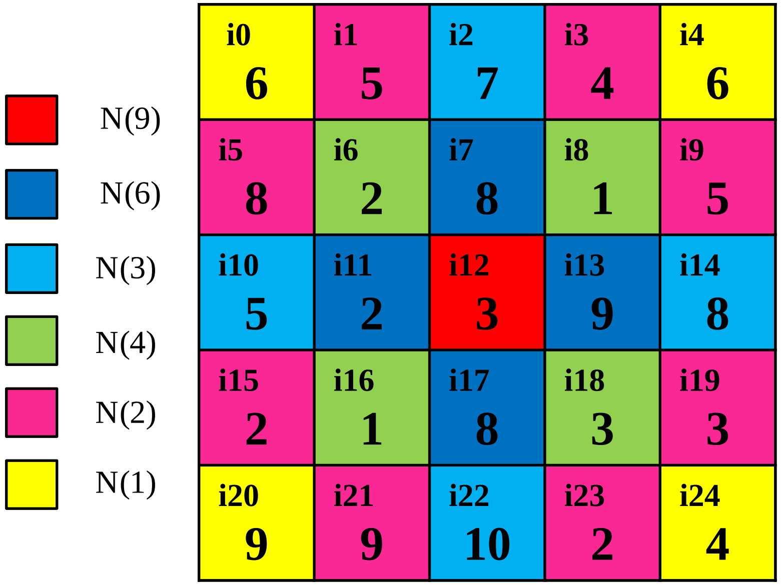
w0 2	w1 2	w2 3
w3 1	w4 2	w5 1
w6 1	w7 3	w8 1

Kernel

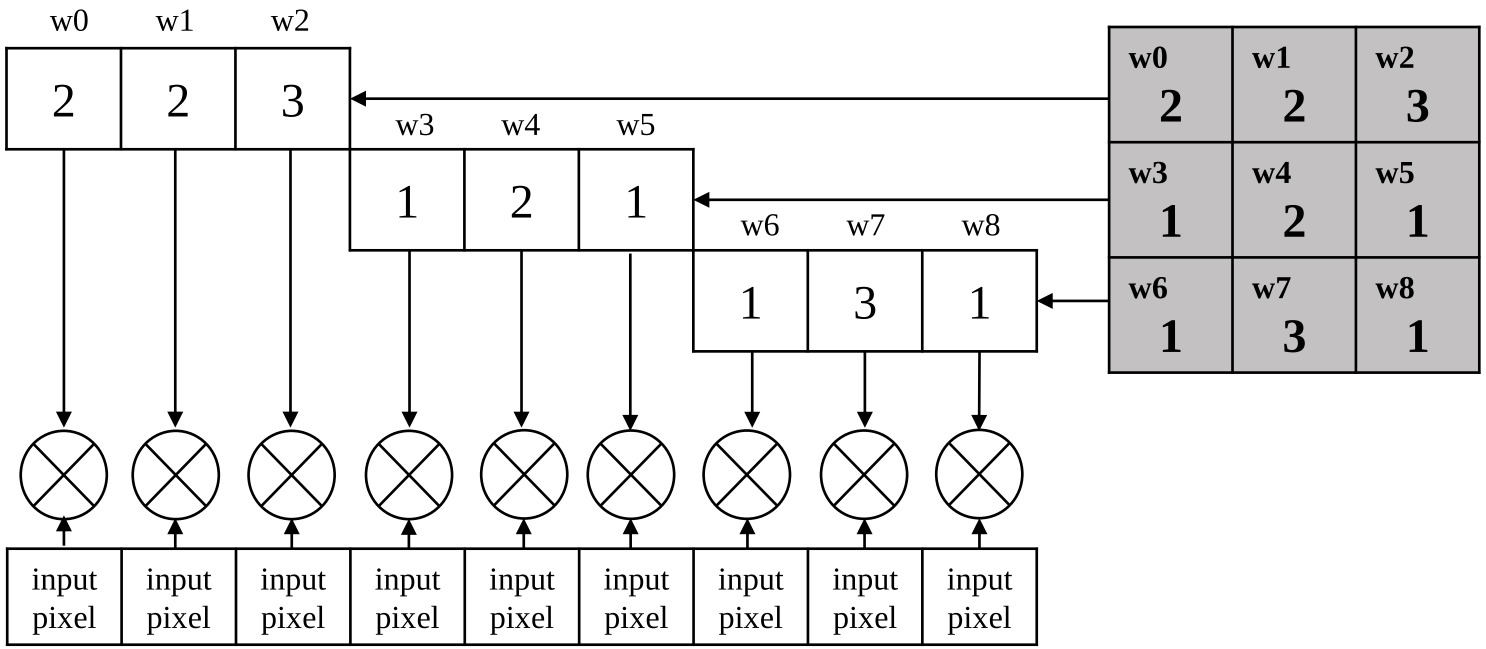
o0 77	o1 75	o2 93
o3 69	o4 68	o5 82
o6 81	o7 98	o8 85

Output

SPP2D Convolution Operation 1



(a) 5 × 5 input example



(b) The kernel is unfolded into an array of multipliers

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**

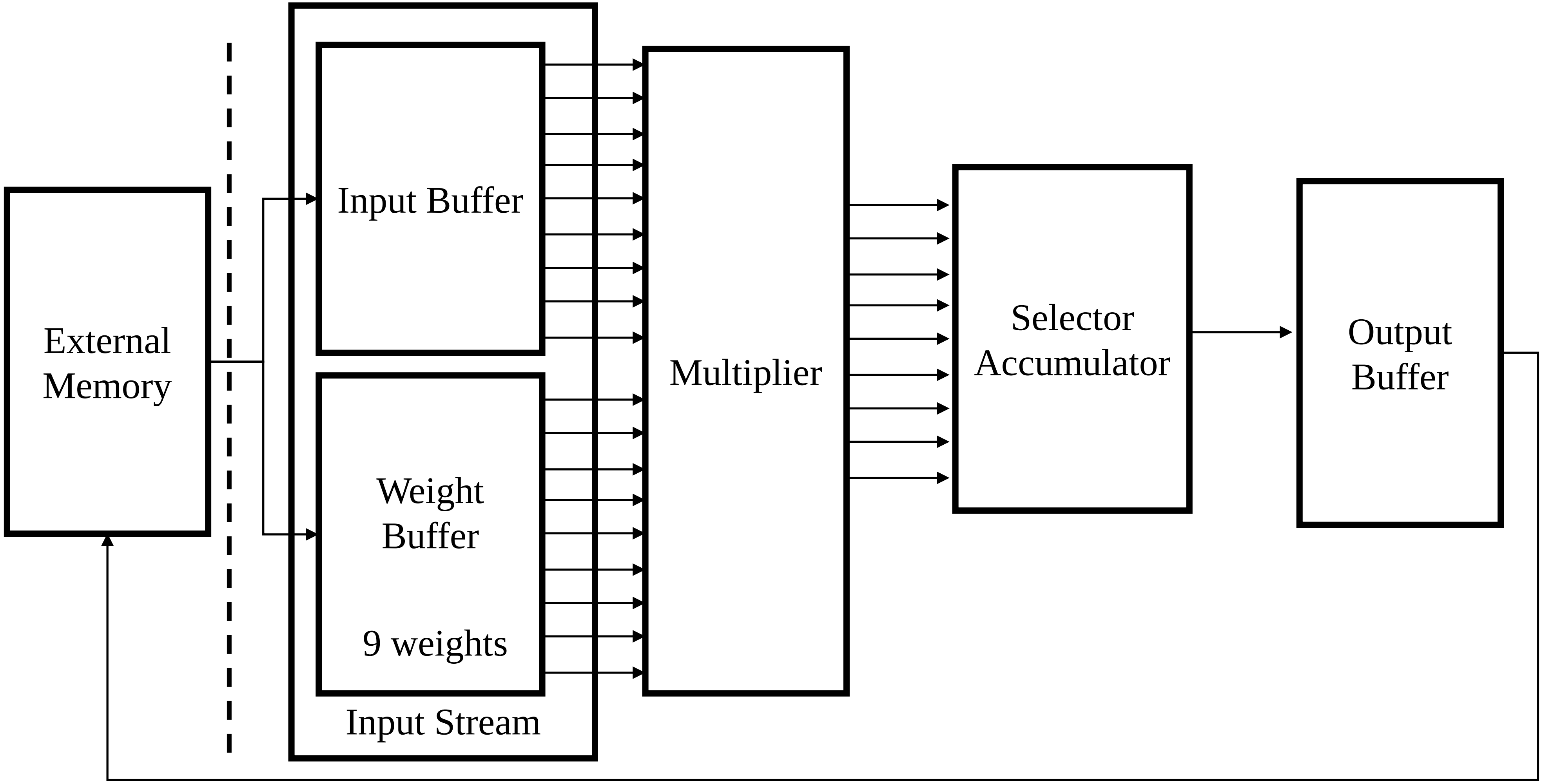
Output

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

Part 3

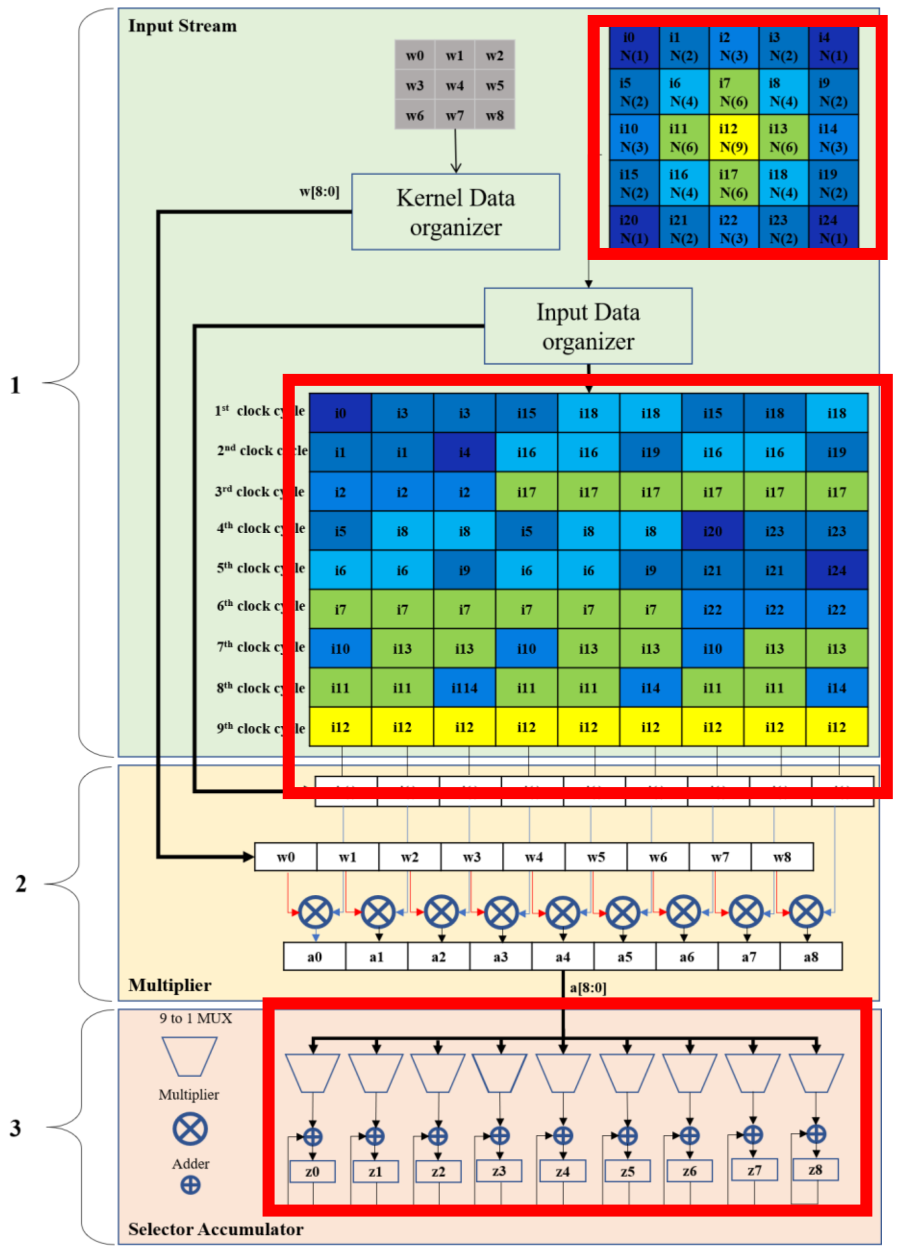
- Architecture proposed in ISCAS 2020
- Results

Hardware Architecture



Hardware Architecture

- Can arrive at 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 is designed for a 5x5 input and 3x3 weights. We need to scale it to an input size of 224x224.



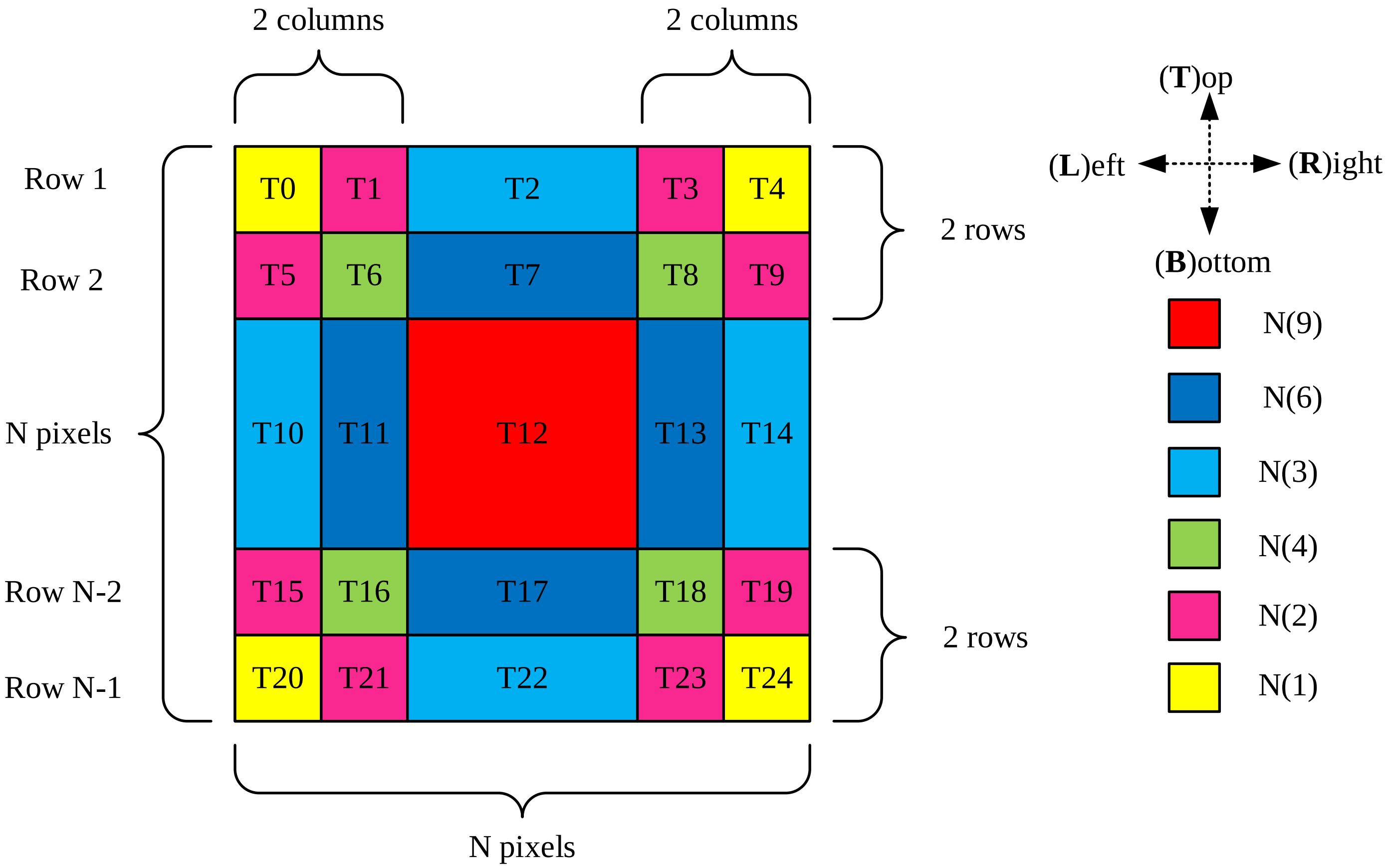
5x5 input results 25 pixels

Would require a big buffer to accommodate 81 pixels

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

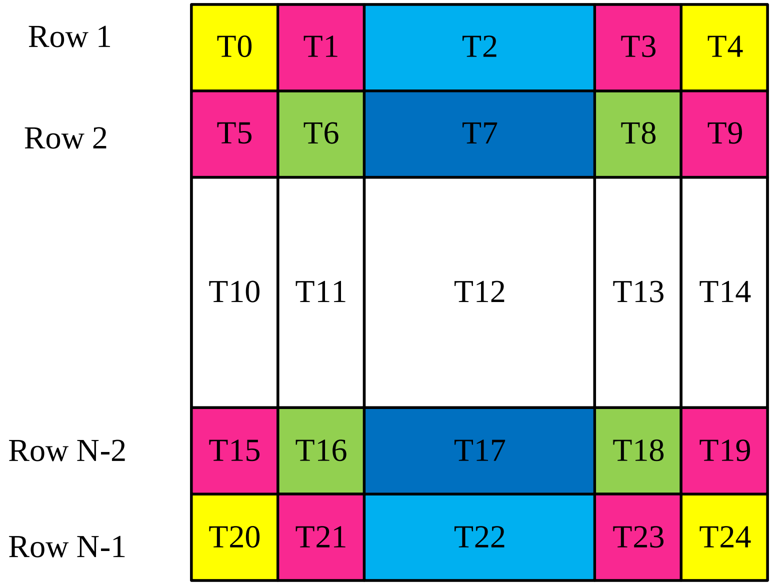
Santa Clara University

Types of Input

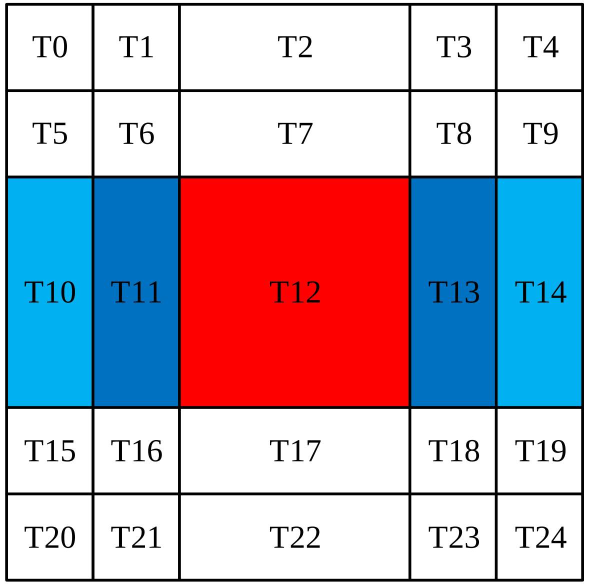


The complementary types of pixels are T12, (T2,T17), (T7, T22), (T10, T13), (T11, T14), (T6, T9), (T21, T24), (T5, T8, T20, T23), (T1, T4, T16, T19), (T0, T3, T15, T18).

Input Stream Organization



Rows 1 and N-2 are complementary and 2 and N-1 are complementary



Rows 3 to N-3 are most of input and are sent after rows 1,2, N-2, N-1

- Complementary Sets in Row 1 and N-2: (i17,i2), (i8,i5,i23,i20), (i18,i3,i15,i0)
- Complementary Sets in Row 2 and N-1: (i7,i22), (i6,i19,i21,i24), (i16,i1,i19,i4), (i13, i10)
- Complementary Sets in Rows 3 to N-3: (i11,i 14),

Input Stream Organization

Row 1	T0	T1	T2	T3	T4
Row 2	T5	T6	T7	T8	T9
	T10	T11	T12	T13	T14
Row N-2	T15	T16	T17	T18	T19
Row N-1	T20	T21	T22	T23	T24

T18+T3+T15+T0

Row 1	T0	T1	T2	T3	T4
Row 2	T5	T6	T7	T8	T9
	T10	T11	T12	T13	T14
Row N-2	T15	T16	T17	T18	T19
Row N-1	T20	T21	T22	T23	T24

T16+T1+T19+T4

Row 1	T0	T1	T2	T3	T4
Row 2	T5	T6	T7	T8	T9
	T10	T11	T12	T13	T14
Row N-2	T15	T16	T17	T18	T19
Row N-1	T20	T21	T22	T23	T24

T2 + T17

Input Stream Organization

Row 1	T0	T1	T2	T3	T4
Row 2	T5	T6	T7	T8	T9
	T10	T11	T12	T13	T14
Row N-2	T15	T16	T17	T18	T19
Row N-1	T20	T21	T22	T23	T24

T8+T5 +T23+T20

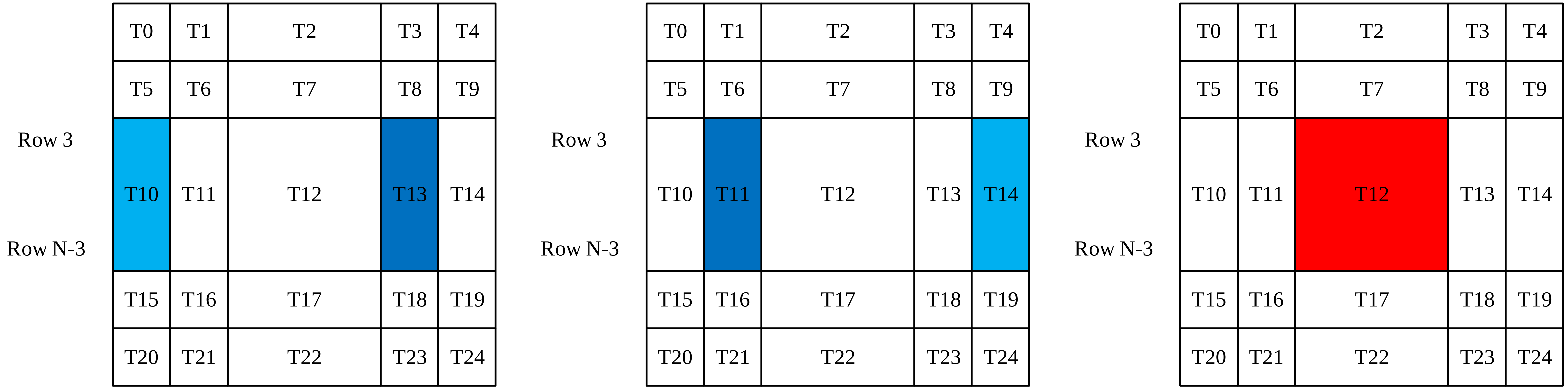
Row 1	T0	T1	T2	T3	T4
Row 2	T5	T6	T7	T8	T9
	T10	T11	T12	T13	T14
Row N-2	T15	T16	T17	T18	T19
Row N-1	T20	T21	T22	T23	T24

T6+T19+T21+T24

Row 1	T0	T1	T2	T3	T4
Row 2	T5	T6	T7	T8	T9
	T10	T11	T12	T13	T14
Row N-2	T15	T16	T17	T18	T19
Row N-1	T20	T21	T22	T23	T24

T7 +T22

Input Stream Organization



T10 +T13　　T11 +T14　　T12

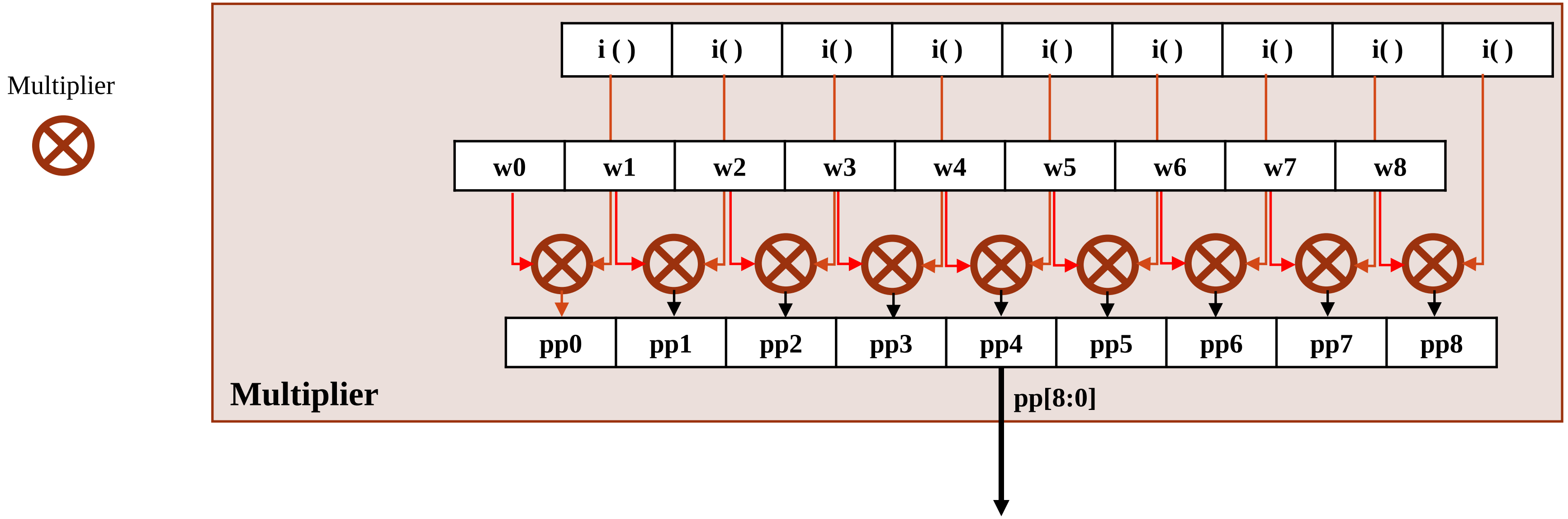
Input Buffer

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)

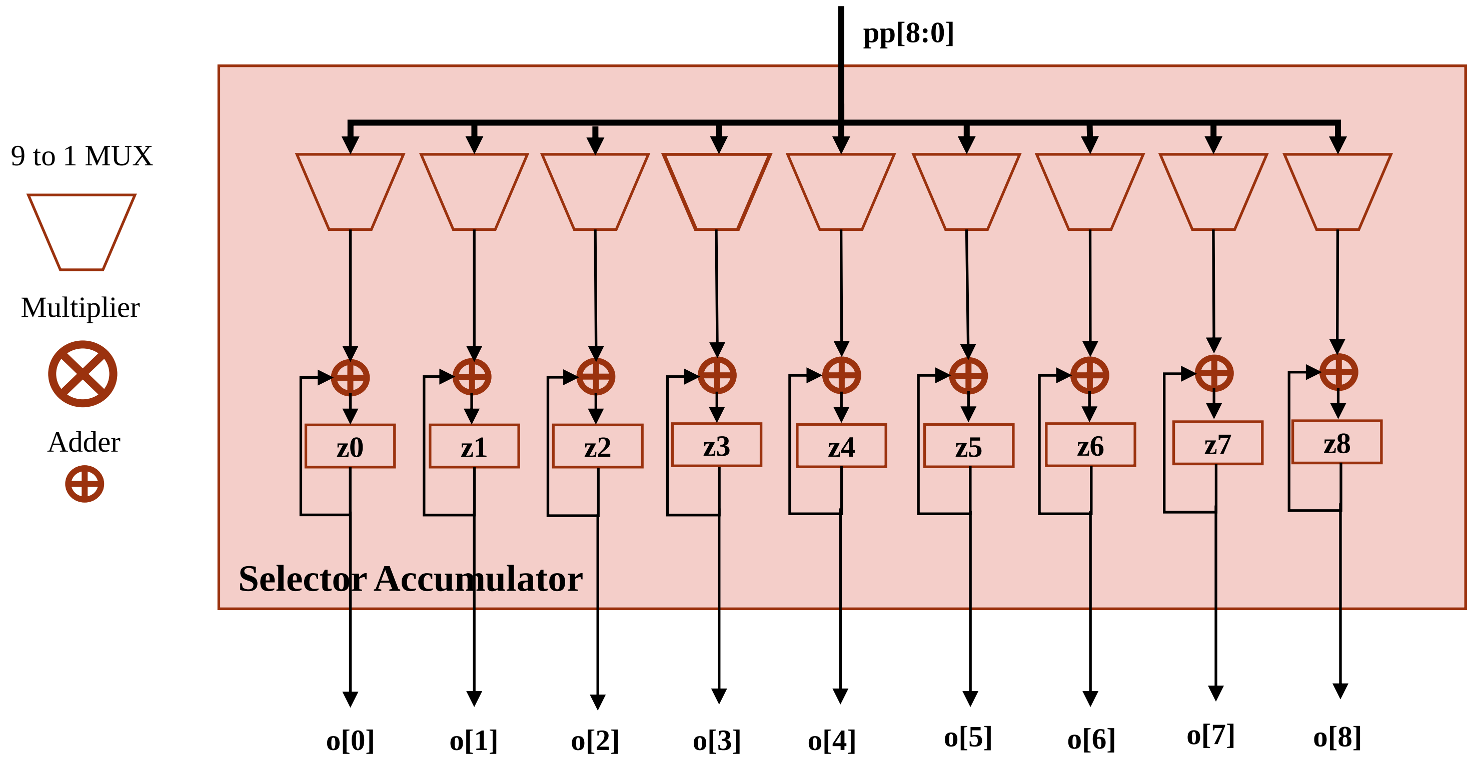
Input

	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		i0	i1	i2	i5	i6	i7	i10	i11	i12
w1		i3	i1	i2	i8	i6	i7	i13	i11	i12
w2		i3	i4	i2	i8	i9	i7	i13	i114	i12
w3		i15	i16	i17	i5	i6	i7	i10	i11	i12
w4		i18	i16	i17	i8	i6	i7	i13	i11	i12
w5		i18	i19	i17	i8	i9	i7	i13	i14	i12
w6		i15	i16	i17	i20	i21	i22	i10	i11	i12
w7		i18	i16	i17	i23	i21	i22	i13	i11	i12
w8		i18	i19	i17	i23	i24	i22	i13	i14	i12

Multiplication Operation



Selector Accumulator



Selector Accumulator

i18+i3+i15+i0	w0i0	w1i3	w2i3	w3i15	w4i18	w5i18	w6i15	w7i18	w8i18	1cc
i16+i1+i19+i4	w0i1	w1i1	w2i4	w3i16	w4i16	w5i19	w6i16	w7i16	w8i19	2cc
i17 +i2	w0i2	w1i2	w2i2	w3i17	w4i17	w5i17	w6i17	w7i17	w8i17	3cc
i8+i5 +i23+i20	w0i5	w1i8	w2i8	w3i5	w4i8	w5i8	w6i20	w7i23	w8i23	4cc
i6+i19+i21+i24	w0i6	w1i6	w2i9	w3i6	w4i6	w5i9	w6i21	w7i21	w8i24	5cc
i7 +i22	w0i7	w1i7	w2i7	w3i7	w4i7	w5i7	w6i22	w7i22	w8i22	6cc
i13 + i10	w0i10	w1i13	w2i13	w3i10	w4i13	w5i13	w6i10	w7i13	w8i13	7cc
i11 +i14	w0i11	w1i11	w2i14	w3i11	w4i11	w5i14	w6i11	w7i11	w8i14	8cc
i12	w0i12	w1i12	w2i12	w3i12	w4i12	w5i12	w6i12	w7i12	w8i12	9cc
	w0	w1	w2	w3	w4	w5	w6	w7	w8	

Results

	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 existing* implementation

* 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.

Conclusion

- SPP2D is a fast convolution technique
- It avoid repetitive reads of input pixels.
- The proof of concept is very promising and can be used accelerate networks of higher sizes

Question and Answer

Thankyou for your time.