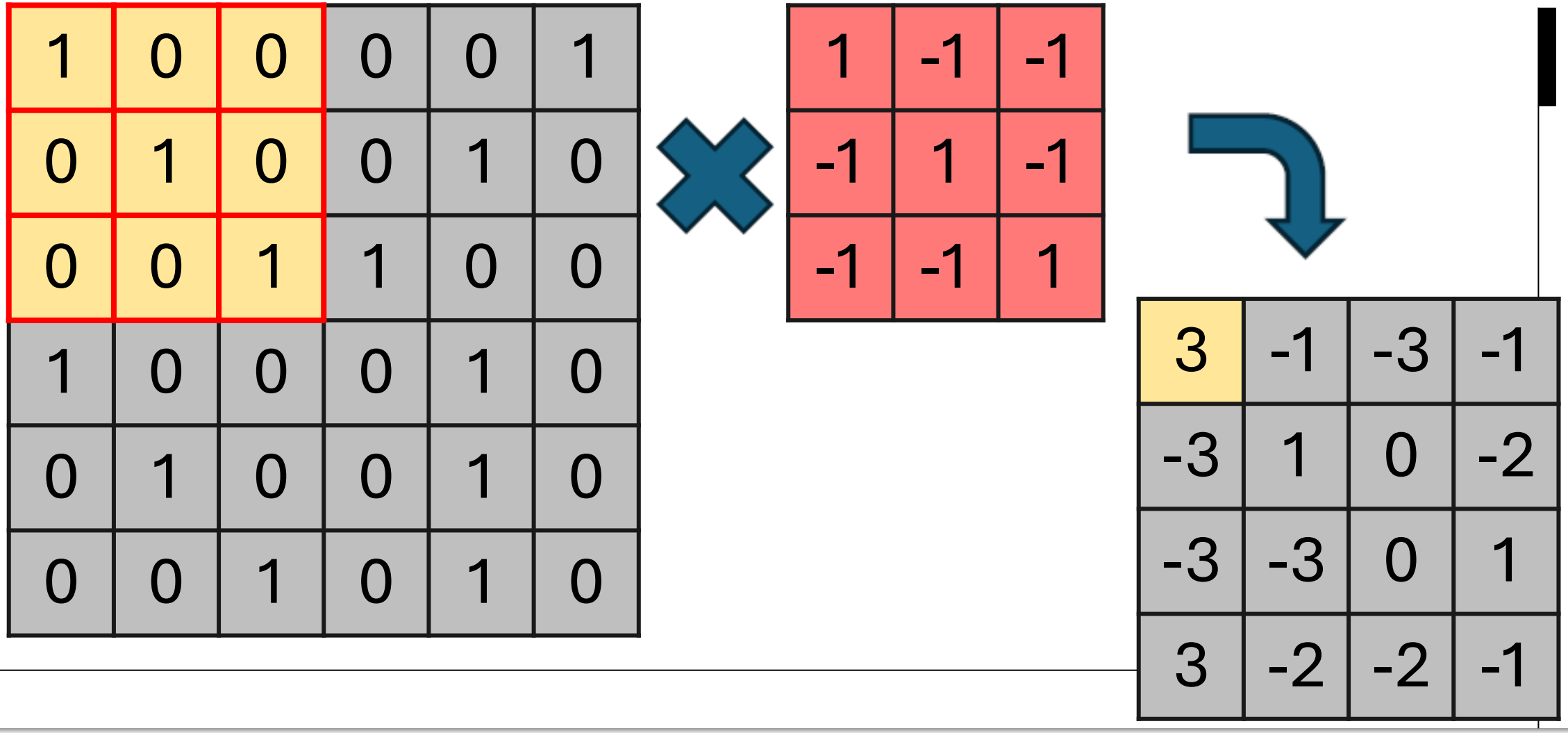
HDL homework 5

王鼎睿 rueimail4work@g-mail.nsysu.edu.tw

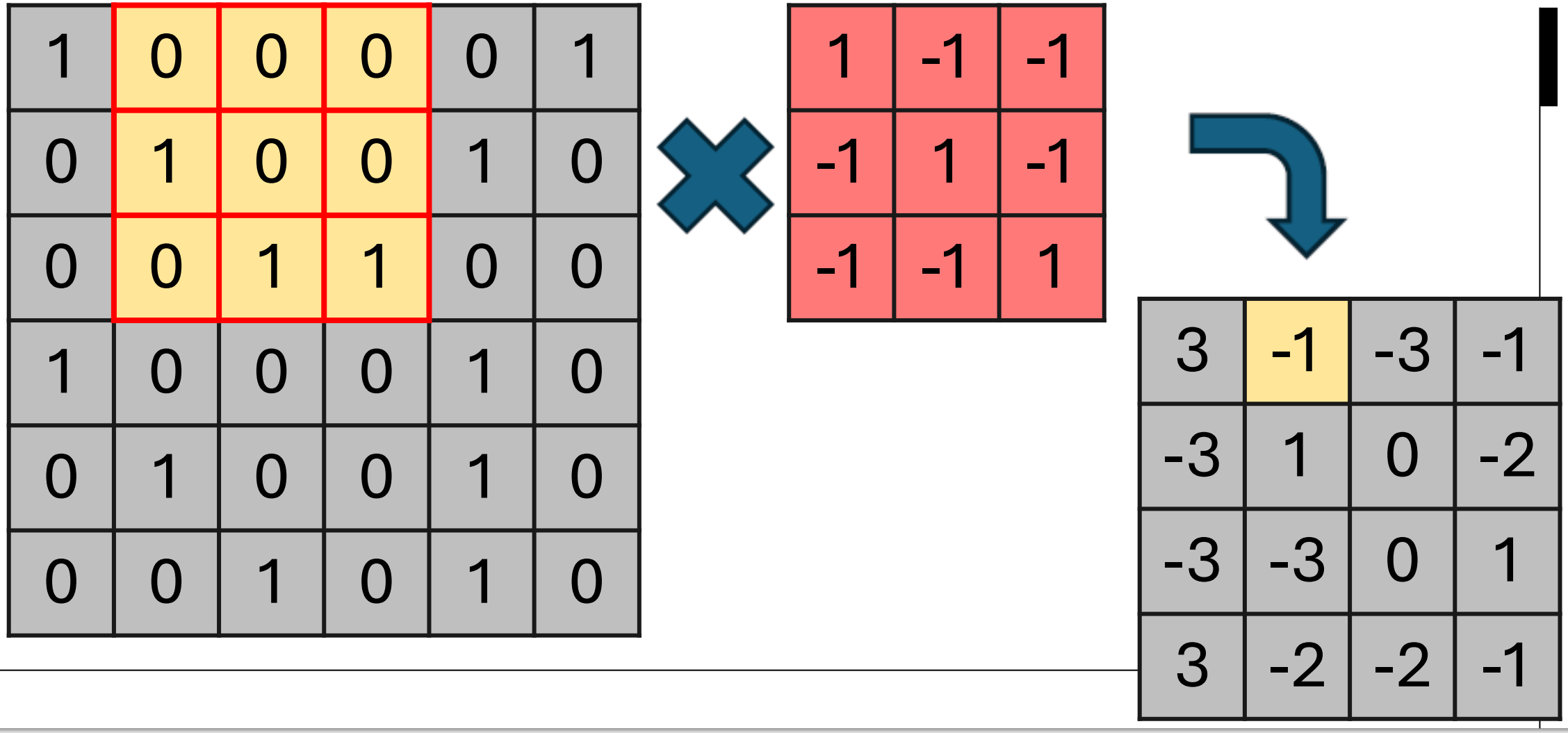
Outline

- Convolution
- Hardware design
- Bias / Kernel data
- ULP、Padding、data bit width

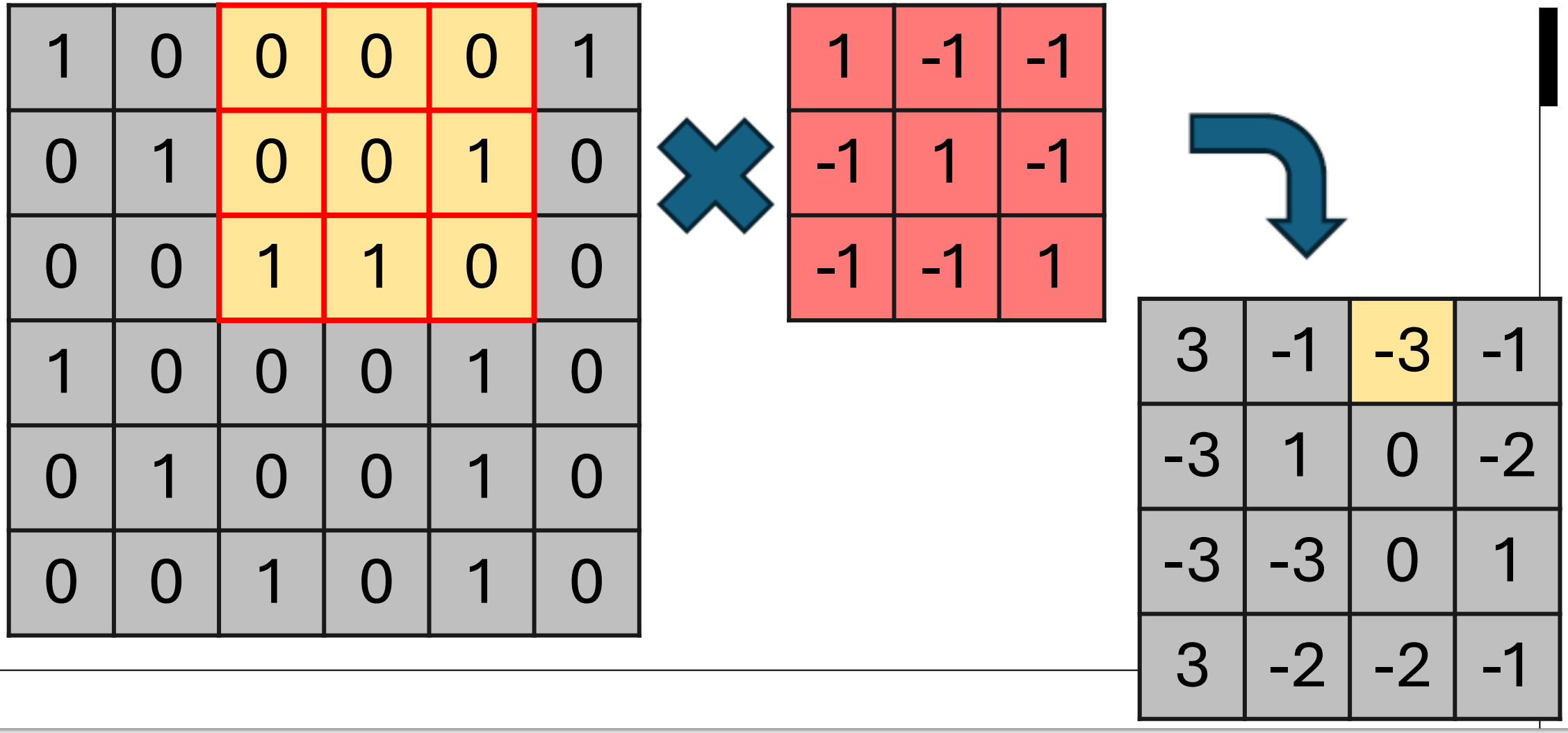
Convolution



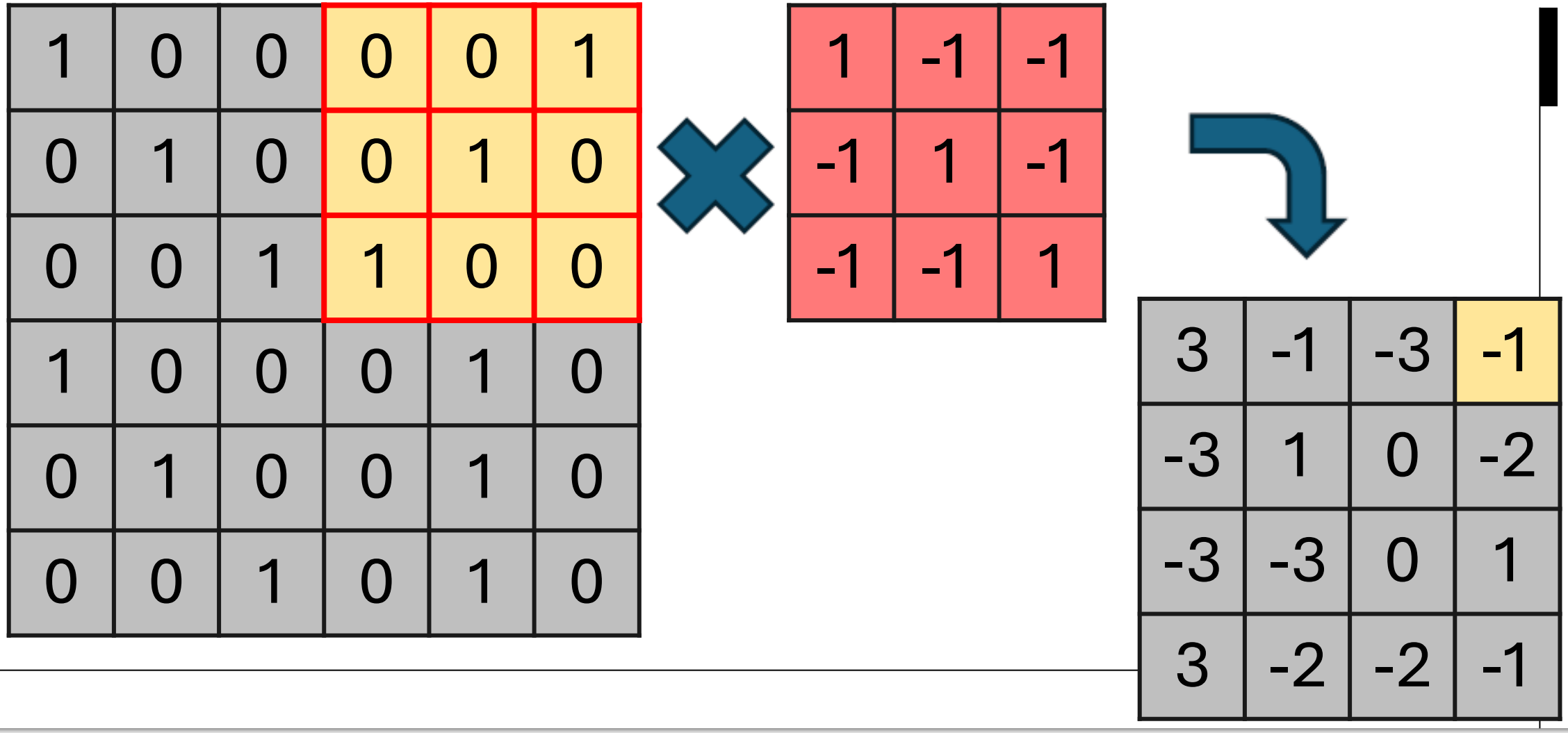
Convolution



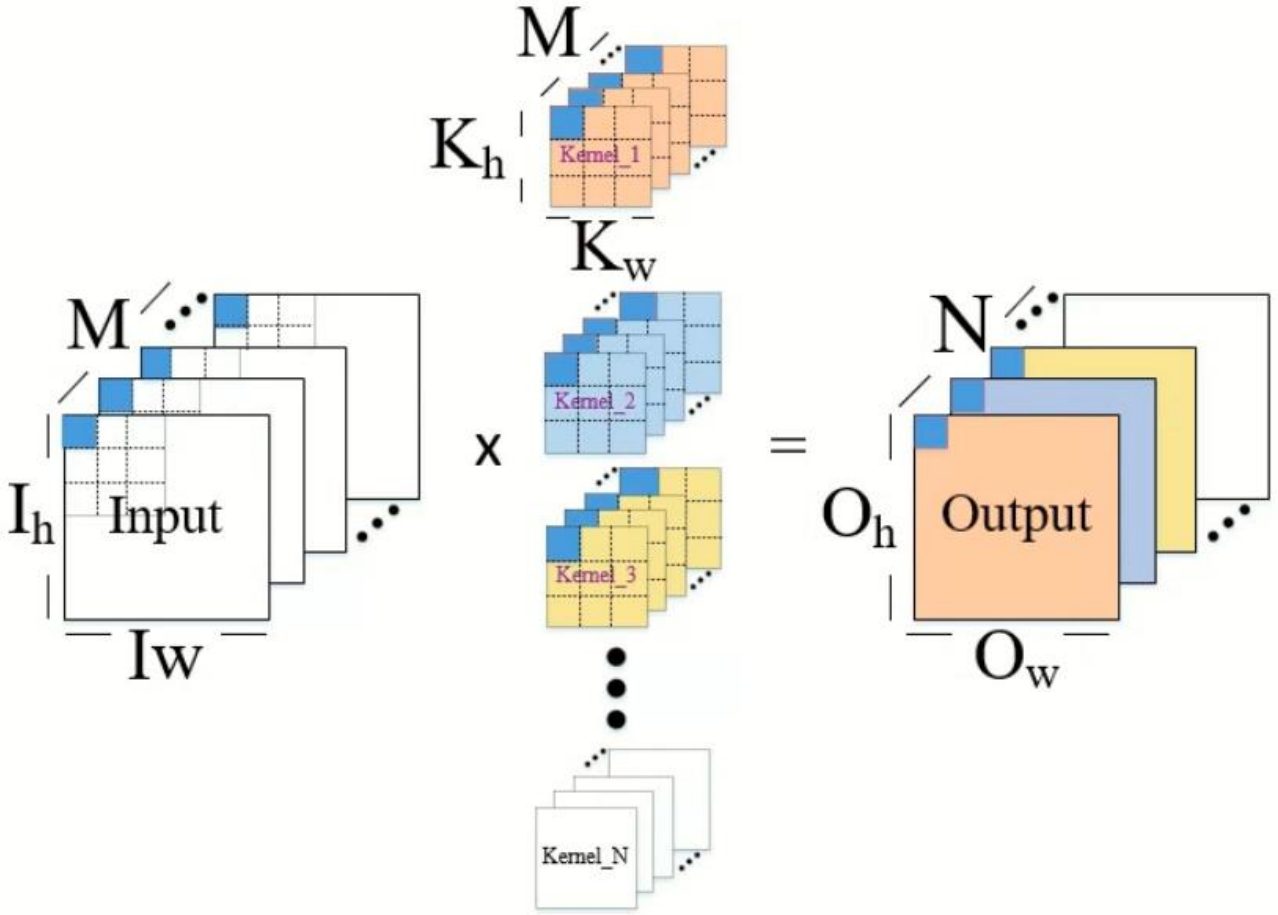
Convolution



Convolution

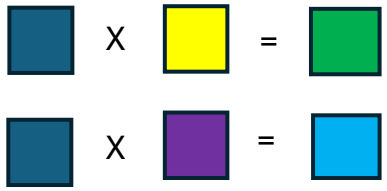


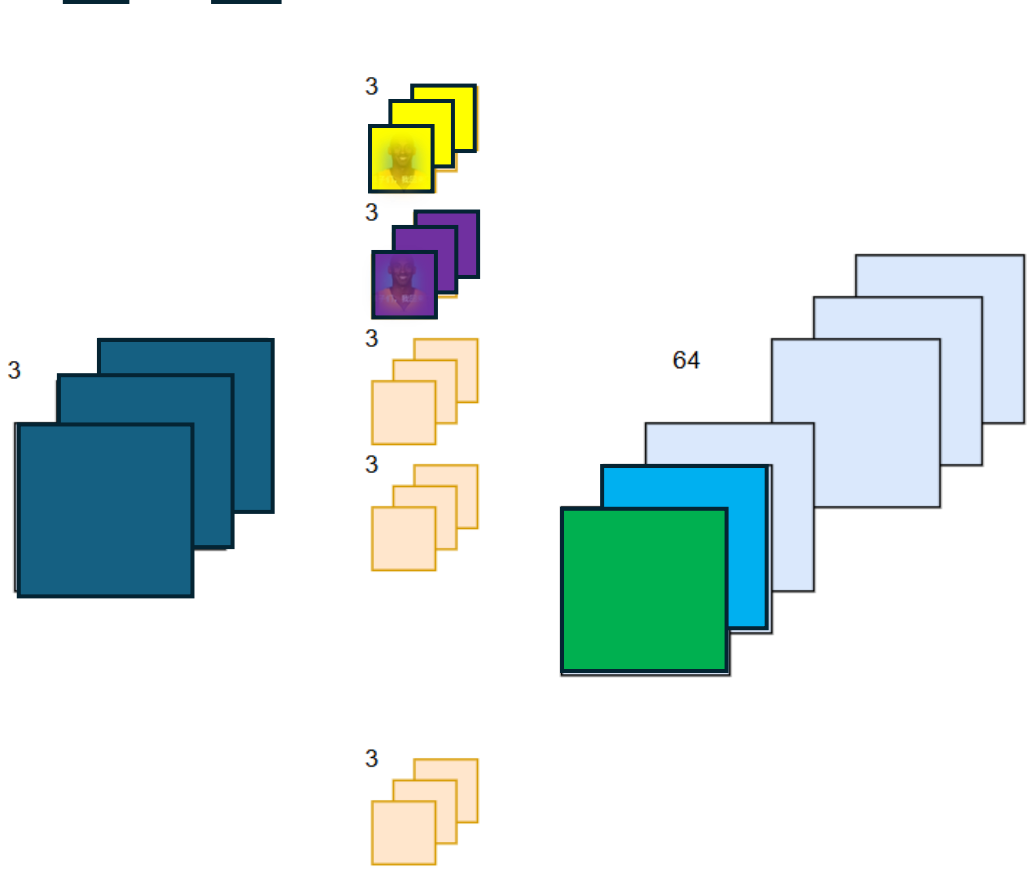
Convolution layer(1/3)



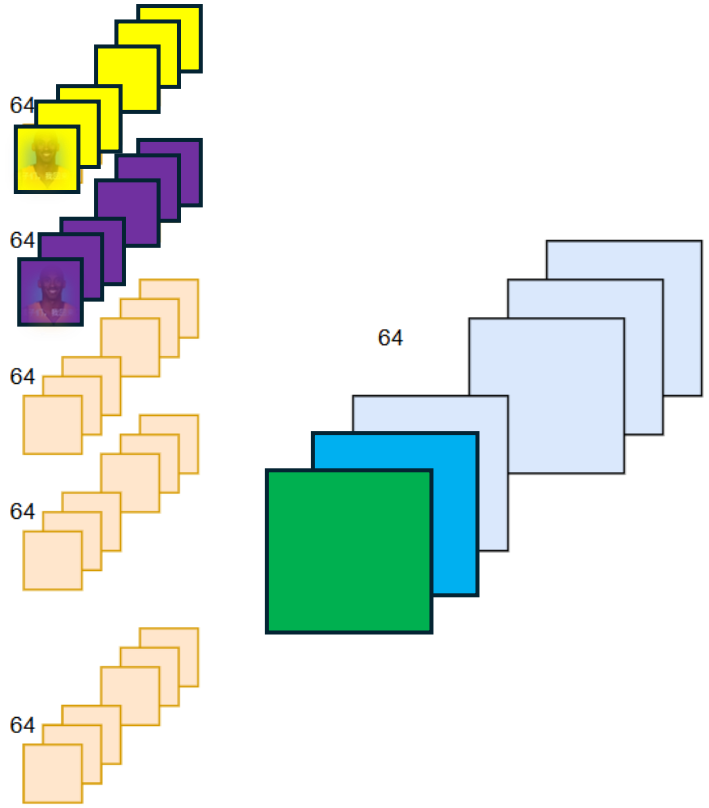
I_h = Input height
I_w = Input width
M = Input Channel
K_h = Kernel height
K_w = Kernel width
O_h = Output height
O_w = Output width
N = Output Channel

Convolution layer(2/3)



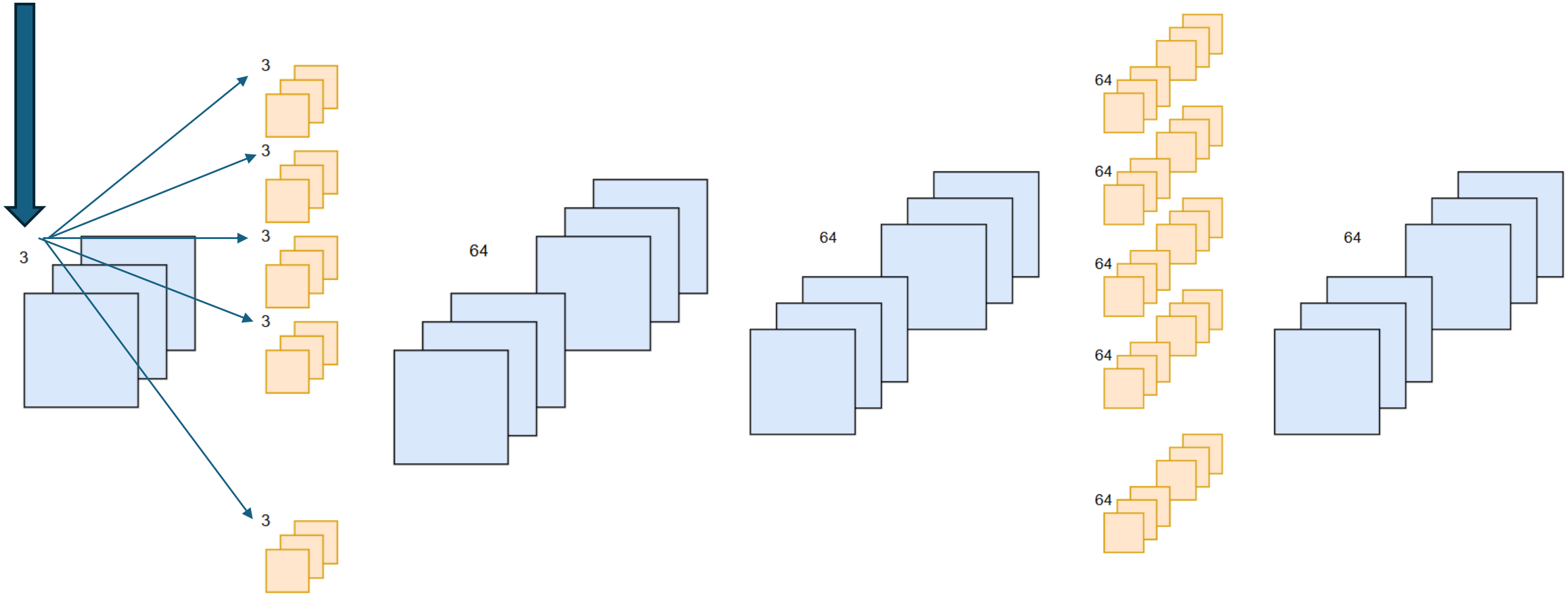


(a)First layer of VGG-16.



(b)Second layer of VGG-16.

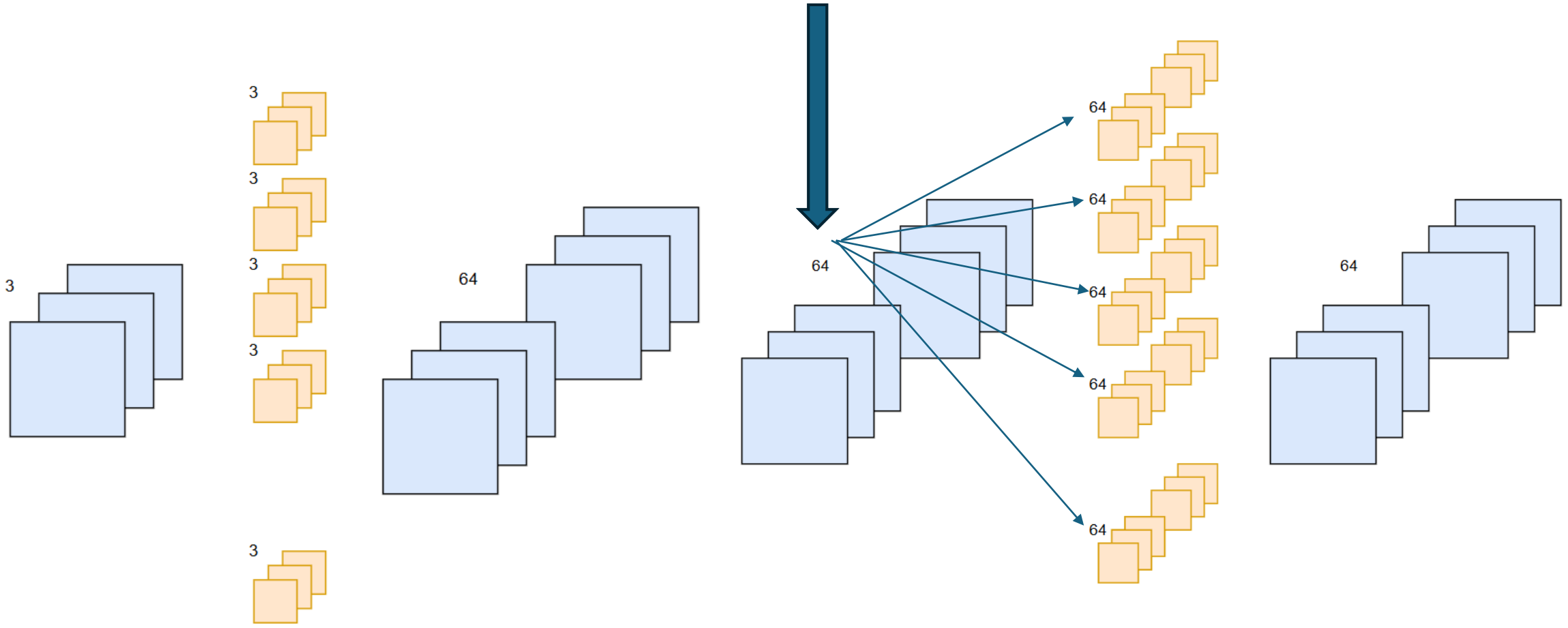
Convolution layer(2/3)



(a)First layer of VGG-16.

(b)Second layer of VGG-16.

Convolution layer(2/3)

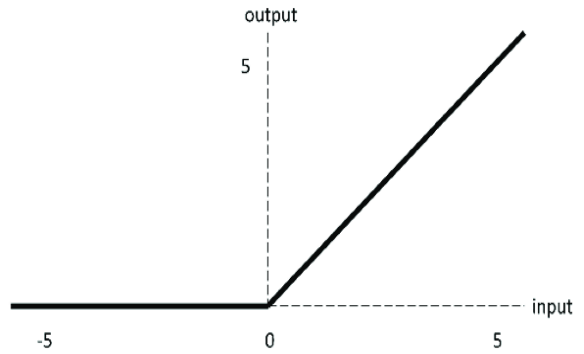


(a)First layer of VGG-16.

(b)Second layer of VGG-16.

Convolution layer(3/3)

- 正確運算流程

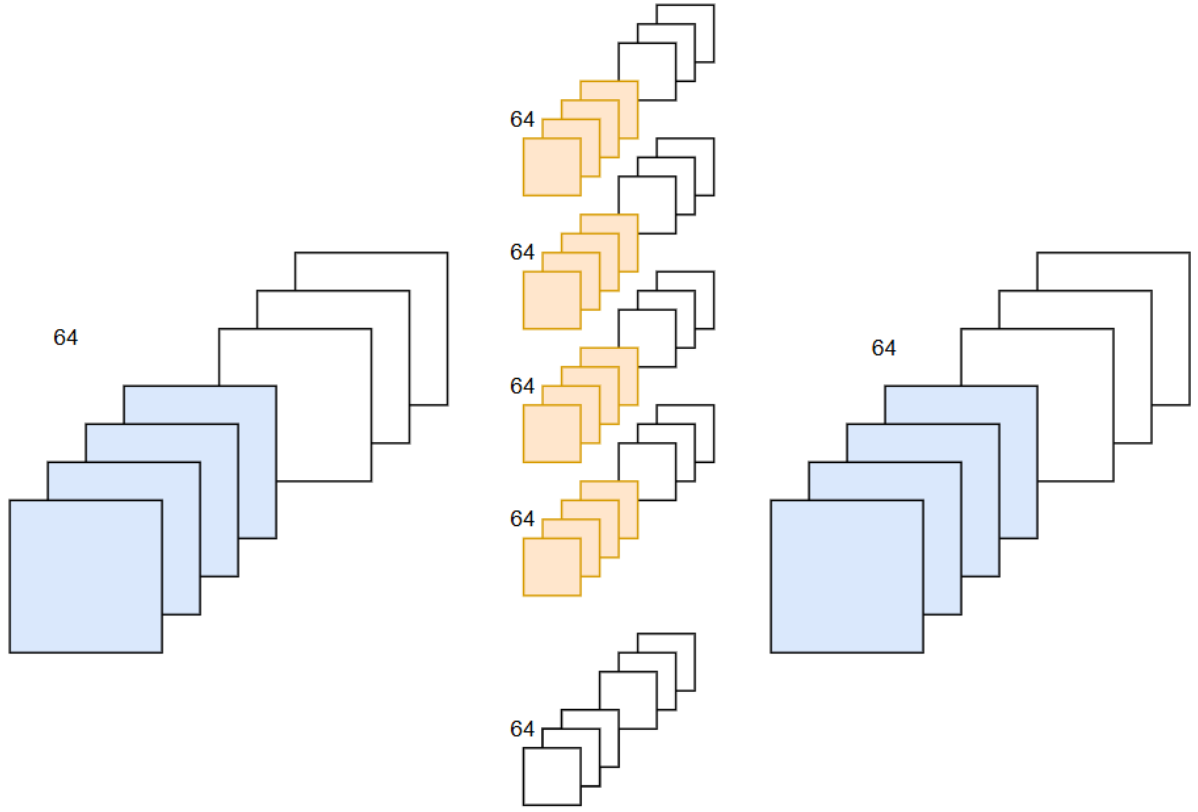


硬體設計(1/4)

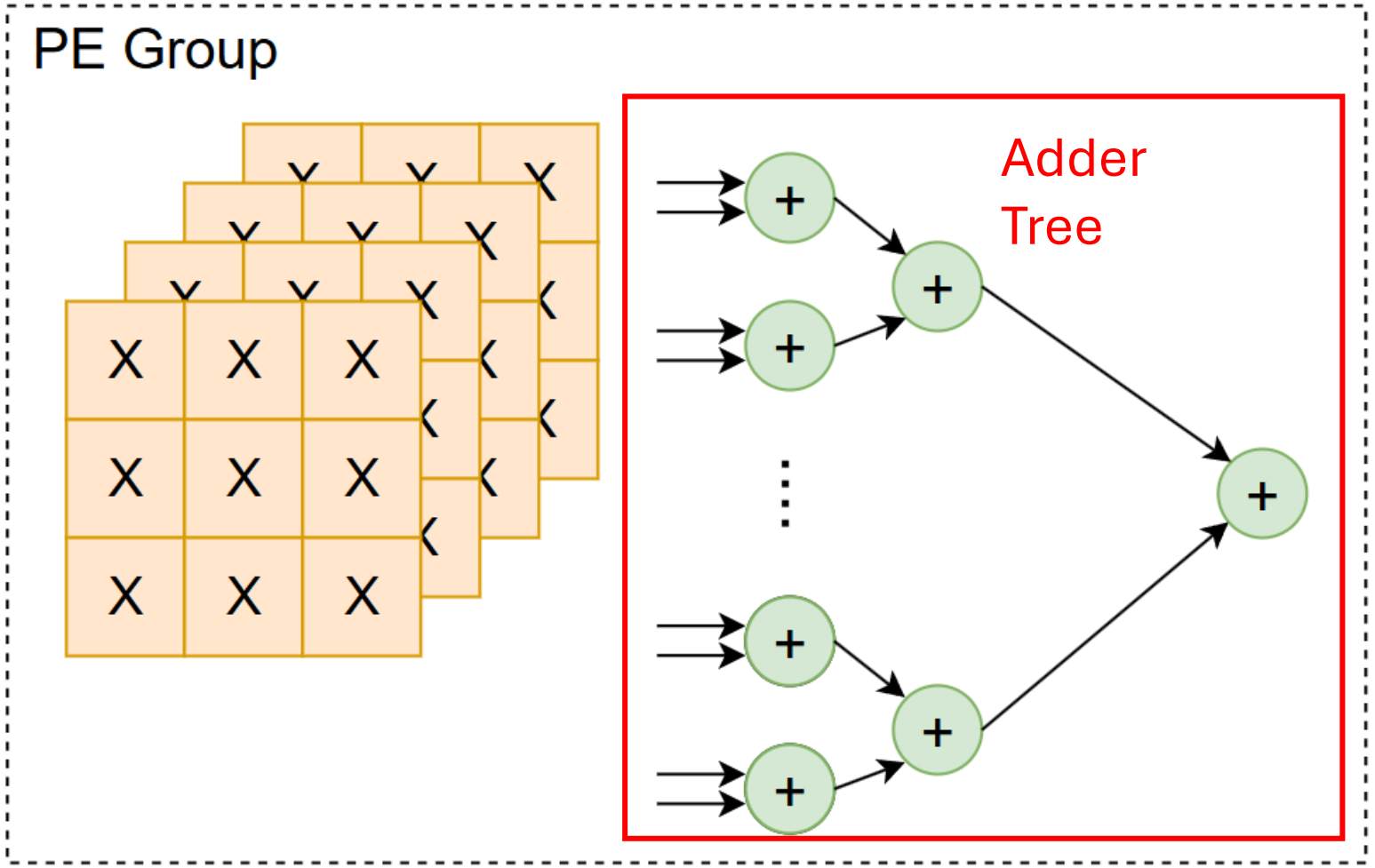
ICP = Input Channel Parallelism = 4

OCP = Output Channel Parallelism = 4

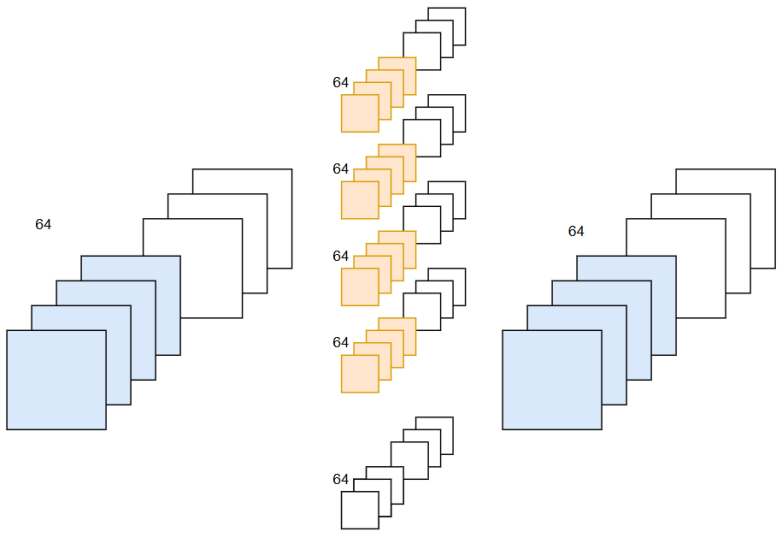
KWP = Kernel Window Parallelism = 9

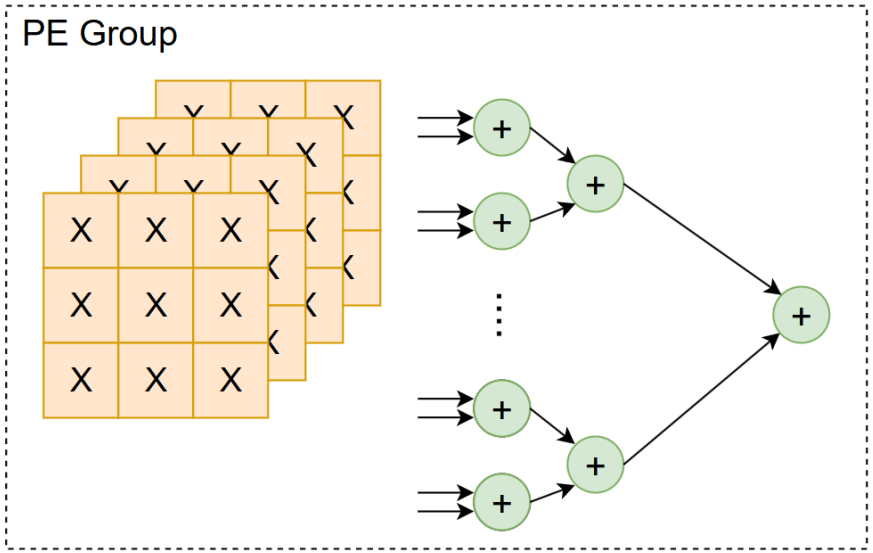


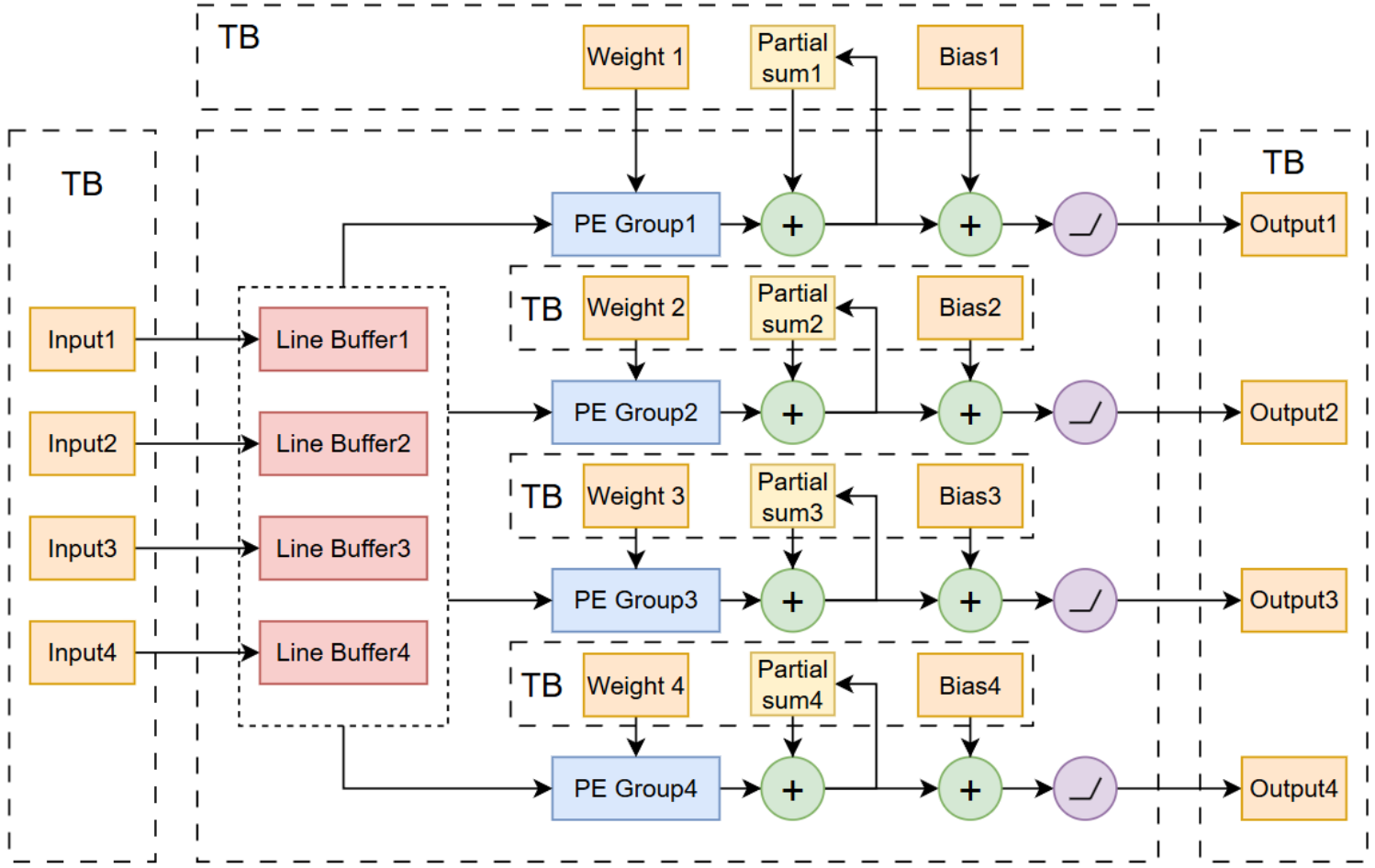
硬體設計(2/4)



硬體設計 (3/4)

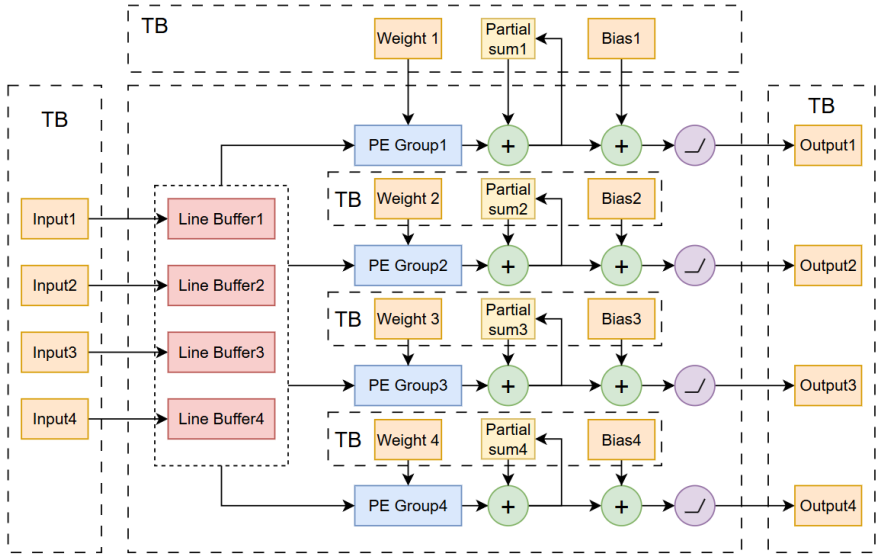

64
64
64
64
64
64


PE Group

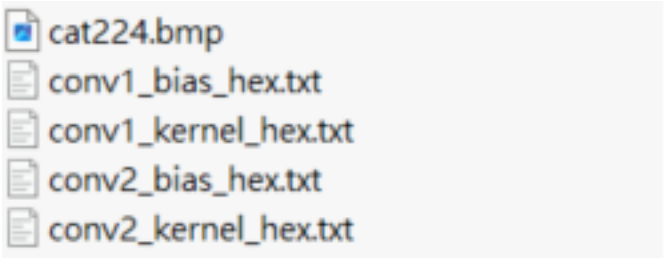

TB
Weight 1
Partial sum1
Bias1
TB
TB
PE Group1
Output1
Input1
Line Buffer1
TB
Weight 2
Partial sum2
Bias2
Input2
Line Buffer2
PE Group2
Output2
TB
Weight 3
Partial sum3
Bias3
Input3
Line Buffer3
PE Group3
Output3
Input4
Line Buffer4
TB
Weight 4
Partial sum4
Bias4
PE Group4
Output4

硬體設計(4/4)

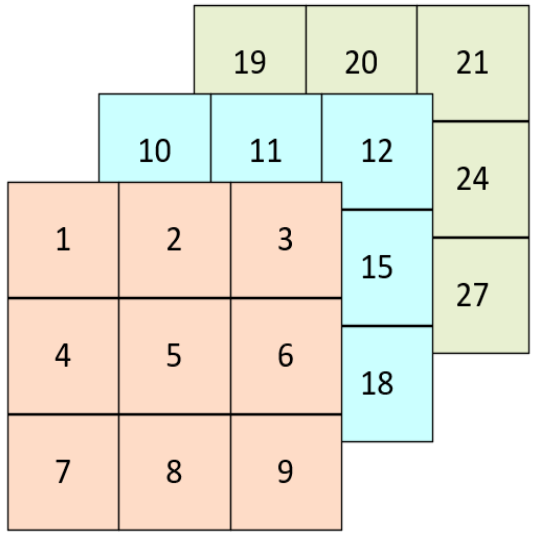
1. 右圖硬體由4個Line Buffer同時給值給4組PE group。每組PE group由4個Channel 的PE組成，每個PE由9個乘法器組成。乘法器個數為4*4*9 = 144個。
2. 運算VGG-16第一層可以同時運算三張input featuremap，(其中一個Line Buffer 用不到)，並且同時得到4張output featuremap，最終整層算完可以得到64張output featuremap。
3. 運算VGG-16第二層可以同時運算4個Channel input featuremap，並且同時得到4個Channel output featuremap的partial sum，最終整層算完可以得到64張output featuremap。
4. 此圖中每組PE group都同時收到4個line Buffer的值，圖中接線僅供參考

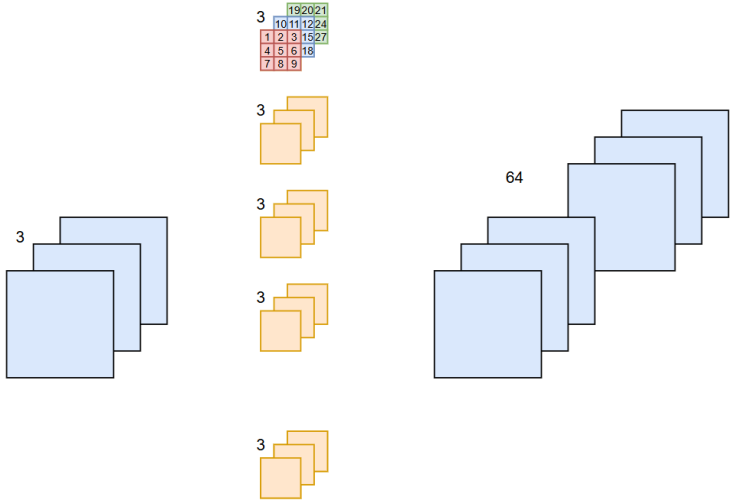


Bias data 、Kernel data 資料擺放順序



EX:conv1_kernal_hex.txt
第1-9筆值為VGG-16第一層第一個kernal的第一個channel的參數。
第10-18筆值為VGG-16第一層第一個kernal的第二個channel的參數。
依此類推其他bias、kernal的檔案。






```
1   001b
2   0017
3   fffd
4   0011
5   0002
6   ffe9
7   fffd
8   fff0
9   ffea
10  0023
11  001c
12  fffb
13  0016
14  0002
15  ffe3
16  fffd
17  ffeb
18  ffe1
19  001e
20  001a
21  fffc
22  0013
23  0003
24  ffe7
25  fffd
26  ffee
27  ffe6
28  0007
```

ULP

累加完，將輸出結果寫入成bmp圖片，需將位元數擷取為8 bit，兩層擷取的位置為下表所示，也可自行調整擷取位置，只要輸出看得出是貓即可。

	ULP
Layer 1	[11 : 4]
Layer 2	[14 : 7]

(**參考) Lyr1 & Lyr2 Padding 在TB做 (8+1bit)

Lyr1 輸入資料最高位元補0(sign bit)，並在testbench進行Padding 補0

```
for(j= 0; j<`img_size; j=j+1) begin //0~172799
    lyr1_temp[j+oo*50176] = 1'd0,img_data[j*3 + offset + i]};
#2;
end
```

```
if( (h==0) || (h==225) || (w==0) || (w==225))  begin
    pad_Y[(h*226)+w+(i*51076)] = 9'd0;
end
else begin
    pad_Y[(h*226)+w+(i*51076)] = lyr1_temp[counter];
    counter = counter+1;
end
#(`period);
```

同理Lyr2 輸入資料為上一層輸出擷取的8位元，並在最高位元補0，且testbench進行Padding 補0

```
if( (h==0) || (h==225) || (w==0) || (w==225))  begin
    pad_Y[(h*226)+w+(i*51076)] = 9'd0;
end
else begin
    pad_Y[(h*226)+w+(i*51076)] = {1'd0,answer[counter]};
    counter = counter+1;
end
#(`period);
```

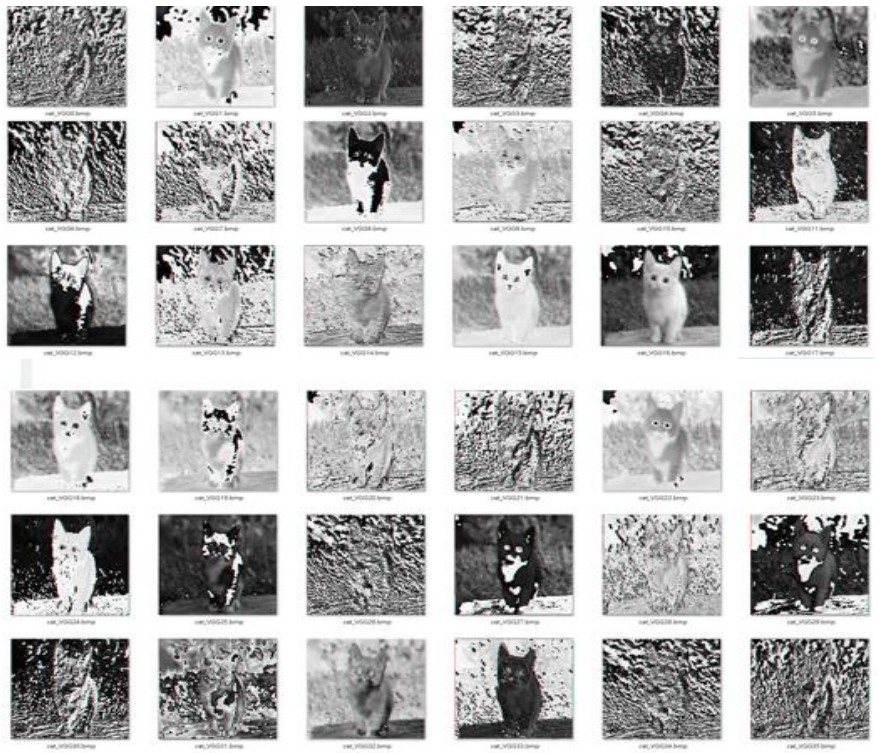
(**參考) Data bit數

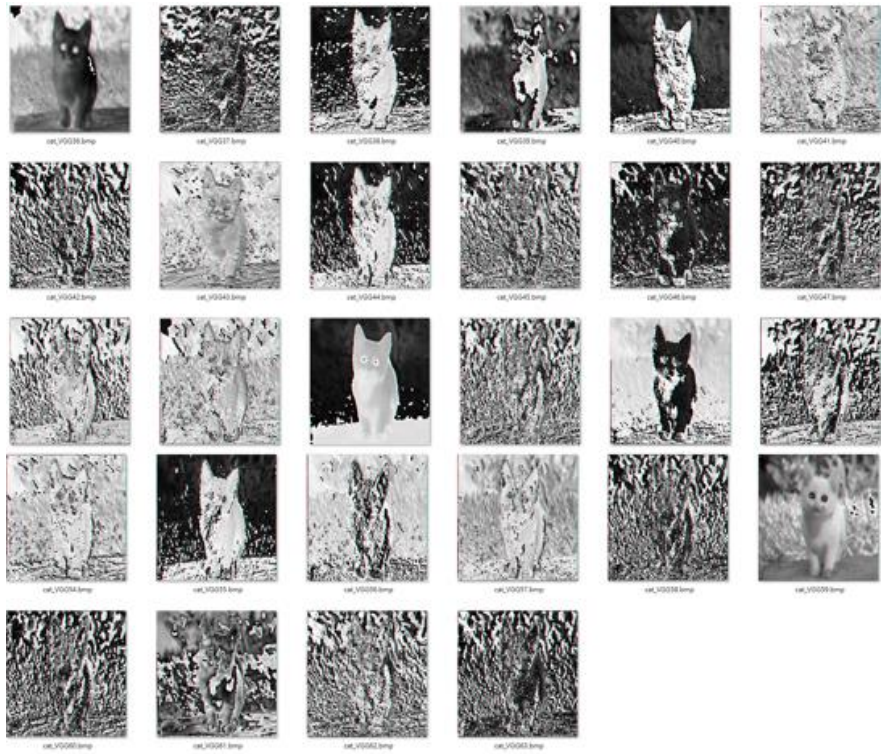
- Input Data(含padding與signed bit): 9bit
- Weight Data : 16bit
- Bias Data : 16bit
- Output Data : 36 bit = (9+16)+$\log_2 36$ + $\log_2 16$ + 1
- Write to BMP : 8bit

```
reg  signed   [8:0]   input_data  ;
reg  signed   [15:0]  weight       [0
reg  signed   [15:0]  bias         [0
wire signed   [35:0]  output_result;
reg  signed   [7:0]  bmp_lyr1_answer
```

固定ULP後圖片結果 - layer 1 (截圖僅供參考)

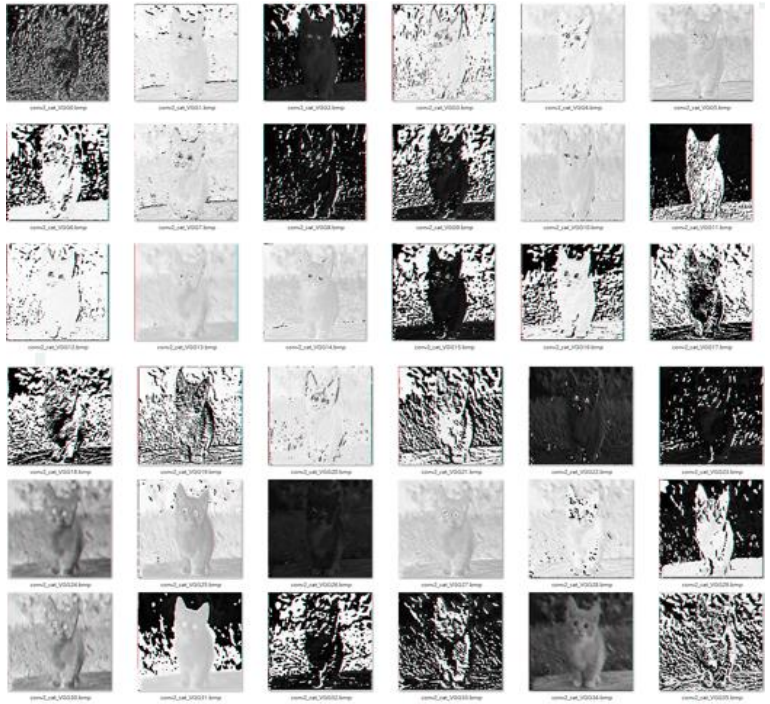
繳交圖片檔請分第一層輸出(64張)，以及第二層輸出(64張)，分別放在名為layer1以及layer2的資料夾。

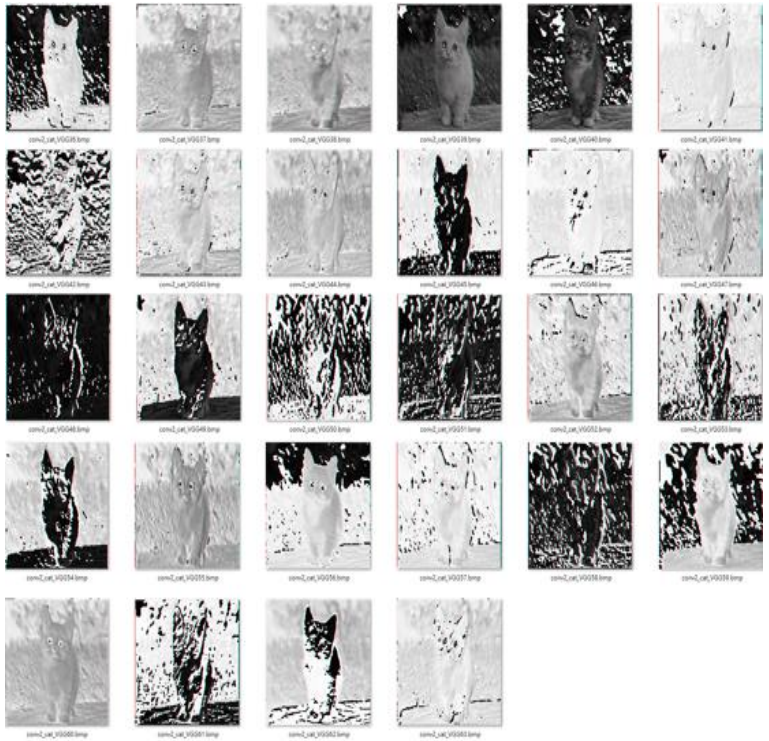




固定ULP後圖片結果 - layer 2 (截圖僅供參考)

繳交圖片檔請分第一層輸出(64張)，以及第二層輸出(64張)，分別放在名為layer1以及layer2的資料夾。





繳交檔案

1. Testbench (20%)
2. Verilog RTL code & Gate Level code
   -LineBuffer (5%)
   -PE (5%)
   -AdderTree (5%)
   -ReLU (5%)
3. Image
   Conv1 image *64(15%)
   -Conv2 image *64(15%)
4. Word
   硬體架構圖解釋(10%)
   -Area資訊和critical path資訊，不需要做optimization，合成出來即可(10%)
   -心得(10%)