通用型深度神經網絡加速器

General Deep Neural Network Accelerator

專業領域: 系統組
組　　別: B265
指導教授: 鄭桂忠 教授
組員姓名: 蔡博智、薛耕典、江振源

Abstract

In Deep Convolutional Neural Network (CNN), the light-weight neural network changes the shape and size of the convolution operation. As a result, general CNN accelerators cannot efficiently operate light-weight network models. In this project, we analyze the characteristics of the four types of convolutions, Standard convolution (SC), Depthwise convolution (DWC), Pointwise convolution (PWC), and Fully connect (FC). According to the hardware architecture of the four loops unroll method, we design a hardware architecture, called Unroll Loop-3or4, that can unroll different loops in different types of convolutions. Let the accelerator maintains high PE Utilization when calculating the four types of convolutions. And according to the design of the dataflow and shift register, the SRAM access is 1/15 to 1/3 of the other dataflow. Our accelerator can achieve area efficiency of 32960 (MACS/gate) and energy efficiency of 381.55 (GMACS/W).

摘要

深度卷積神經網絡(CNN)中，輕量化神經網絡改變了卷積運算的形狀和尺寸，使一般CNN加速器無法有效率地運算輕量化網絡模型。本專題透過分析 Standard convolution (SC), Depthwise convolution (DWC), Pointwise convolution (PWC)和 Fully connect (FC) 四種卷積運算的特性，結合迴圈展開的設計架構，我們設計出一種可以根據不同卷積運算展開不同迴圈的硬體架構 (Unroll Loop-3or4)，使加速器在計算四種卷積運算時都維持著高 PE Utilization。且根據數據流與 shift register 的設計，可以減少 SRAM 讀寫量至其他數據流的 1/15 到 1/3。最後我們設計的加速器可以達到，面積效率 32960 (MACS/gate) 與能源效率 381.55 (GMACS/W)。

1. Introduction

1.1 前言

在深度神經網絡(DNN)的領域中，卷積神經網絡(CNN)被廣泛運用於影像識別、視訊分析等任務，而一般 CNN 龐大的運算量與資料量使其無法運用於硬體效能有限的嵌入式裝置 [1]，輕量化網絡利用改變卷積運算的設計，大大減少運算量與資料量而維持影像識別準確度 [2]，但改變卷積運算設計增加了運算複雜度，網絡的形狀限制了平行運算展開的方向，網絡尺寸縮小使資料複用(data reuse)發生變化，進而使計算單元使用效率(PE Utilization)降低，讓一般加速器無法支援或是無法達到預期的優化效果 [3]。

本專題的目標在於設計出通用於 Standard convolution (SC), Depthwise convolution (DWC), Pointwise convolution (PWC)和 Fully connect (FC)的硬體加速器架構，在不同類型的卷積運算、不同卷積核(kernel)大小都有高 PE Utilization，增加運算速度，並最大化 data reuse，降低 SRAM 的讀寫量，降低資料讀寫的功耗。我們引入四循環卷積運算設計技巧 [4]，用於分析 DWC、PWC、FC 運算，並發現沒有共通的迴圈可以做平行化的展開，於是我們結合數據流與硬體架構做設計，跳脫分成 weight 與 pixel SRAM 的框架，分成高、低 bandwidth 的 SRAM，做不同類型卷積運算時將 weight 與 pixel 存在不同的 SRAM，以達成在相同硬體架構下對不同迴圈的做展開的結果，對於不同類型卷積運算都可以有高 PE Utilization，達到高運算速度、低功耗的目的。

1.2 四循環級別卷積運算

一個 SC 運算可以分成四層的迴圈做分析 [4]，如圖 1，Loop-1 做 kernel window 內的乘累加，Loop-2 做 kernel 內 channel 方向的乘累加，Loop-3 為 kernel 在 input feature map 上的滑動，Loop-4 做不同 kernel 的卷積。

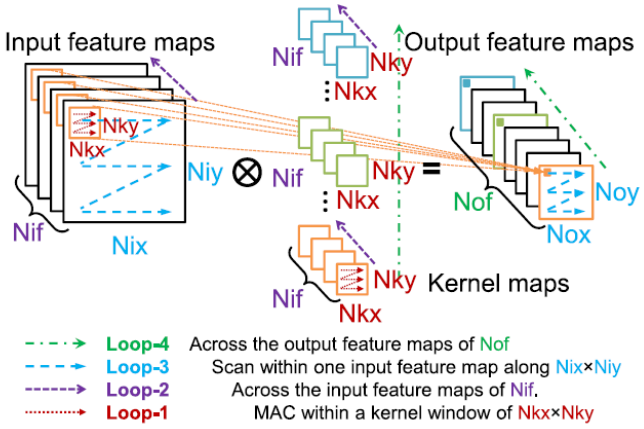

圖 1. 四循環級別卷積運算分析[1]

[1] 圖片來源 [4]

1.3 實現卷積運算設計技巧

實現卷積運算的數據流有三個技巧，分別是 Loop Unroll、Loop Interchange、Loop Tilling [4]，在本專題中我們主要探討 Loop Unroll 與 Loop Interchange。

1.3.1 Loop Unroll: 在做平行運算時，展開不同的迴圈會有不同的平行運算特性，影響 data reuse 和 partial sum 的數量。

1. Unroll loop-1：展開 kernel 的(x, y)方向，在一個 cycle 內不同的 PE 得到同一個 output pixel 的 partial sum，利用 Adder Tree 取和累加。
2. Unroll loop-2：展開 kernel 的 channel 方向，在一個 cycle 內不同的 PE 得到同一個 output pixel 的 partial sum，利用 Adder Tree 取和累加。
3. Unroll loop-3：展開 input feature map 的(x, y)方向，在一個 cycle 內不同的 PE 共用同一個 weight，得到不同 output pixel 的 partial sum。
4. Unroll loop-4：展開多個 kernel，在一個 cycle 內不同的 PE 共用同一個 input pixel，得到不同 output pixel 的 partial sum。

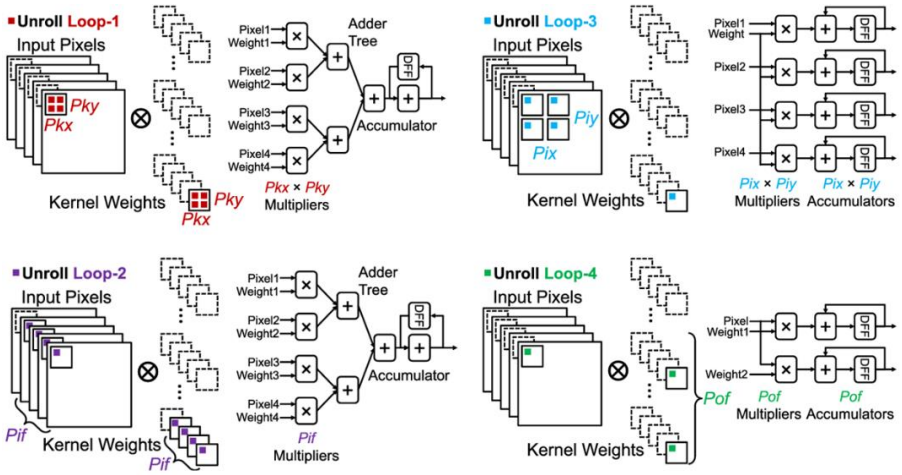

圖 2. Loop Unroll 架構[2]

1.3.2 Loop Interchange：不同迴圈計算的先後順序，會達成不同的數據流，影響資料的讀寫次數。

1. Loop-1 先計算：同一個 PE 在不同的 cycle 計算同一個 output pixel 的 partial sum，可以做到 output stationary (OS)，需要重複讀取 weight 和 input pixel。
2. Loop-2 先計算：同一個 PE 在不同的 cycle 計算同一個 output pixel 的 partial sum，可以做到 output stationary (OS)，需要重複讀取 weight 和 input pixel。
3. Loop-3 先計算：同一個 PE 在不同的 cycle 內用相同的 weight 計算不同的 output pixel，可以做到 weight stationary (WS)，會產生大量的 partial sum 需要存入 SRAM。
4. Loop-4 先計算：同一個 PE 在不同的 cycle 內用相同的 input pixel 計算不同 output pixel，可以做到 input stationary (IS)，會產生大量的 partial sum 需要存入 SRAM。

[2] 圖片來源［4］

2. Research Methodology

2.1 數據流架構設計

2.1.1. 分析不同類型卷積運算

根據 1.2 對 SC 運算的分析方式，我們分析另外三種運算。如圖 3.(b)，我們可以將 FC 視為大小為1 × 1的 input feature map 對多個1 × 1的 kernel 做卷積，因此只有一個 kernel window 內的運算，也沒有 kernel 在 input feature map 上的滑動，所以 FC 只有 Loop-2 和 Loop-4。如圖 3.(c)，PWC 用1 × 1的 kernel 做卷積，同樣只有一個 kernel window 內的運算，所以 PWC 只有 Loop-2，Loop-3，Loop-4。如圖 3.(d)，DWC 只有一個 kernel，所以 DWC 只有 Loop-1，Loop-2，Loop-3，並且 Loop-2 沒有累加。

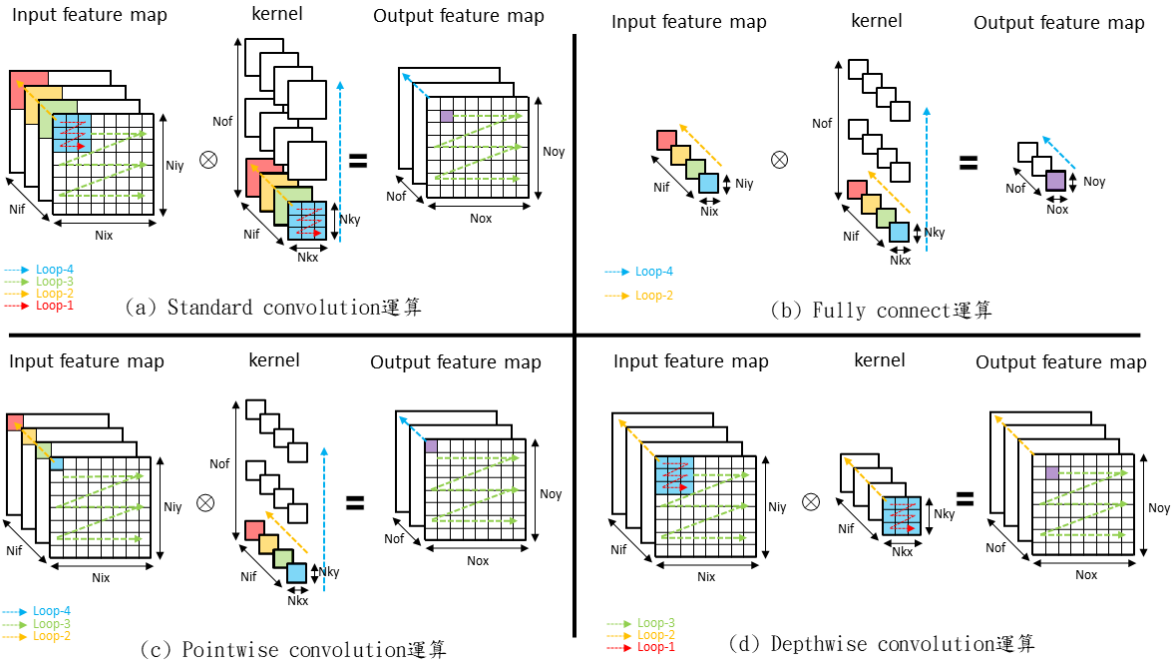

圖 3. 各類卷積運算

表 1. 各類卷積迴圈分析

	loop-1	loop-2	loop-3	loop-4
Standard Convolution	V	V	V	V
Fully Connect		V		V
Pointwise Convolution		V	V	V
Depthwise Convolution	V	O (無累加)	V	

2.1.2. Loop Unroll 設計

加速器為了提高運算速度會展開卷積運算的迴圈做平行化的計算，而不同類型的卷積運算如表 1 所分析，會缺乏不同的迴圈，如果展開的迴圈是該卷積運算所缺乏的，會造成 PE Utilization 降低，運算數度變慢，且需要額外的控制訊號。我們的目標在於設計出通用的硬體架構，在四種不同的卷積運算都有高 PE Utilization，我們先分析四個迴圈的 Loop Unroll 設計，並在最後提出我們設計的 Unroll Loop 方式。

1. Unroll Loop-1:

Unroll Loop-1 的好處在於相同 PE 數量的運算下，利用 Adder Tree 即時的累加，需要較低的 output bandwidth 與較小的暫存器儲存 partial sum。而其面臨的問題在於 kernel 的大小不固定，常見的 kernel 大小為3×3，但仍會有5×5或7×7的 kernel 出現，若是要設計出適應不同 kernel 大小的加速器，需要額外複雜的控制訊號，且在 input pixel 的讀取設計會非常的困難，需要很大的面積代價。若是特化成只能做3×3卷積運算的加速器，還是會面臨 PWC、FC 沒有 Loop-1 的問題，使 PE Utilization 降到原本的 1/9。

2. Unroll Loop-2:

Unroll Loop-2 的好處也是有較低的 output bandwidth 與較小的暫存器。而其面臨的問題是 DWC 的 Loop-2 沒有累加，我們可以選擇降低做 DWC 時的 PE Utilization，或是利用增加 output bandwidth 與增加暫存器數量來維持原本的 PE Utilization，會造成 Unroll Loop-2 的好處消失，但仍可以增加運算的平行度。

3. Unroll Loop-3:

Unroll Loop-3 的好處在於有 weight 的 spatial reuse，可以降低 weight bandwidth，減少 weight 的讀取次數，且拿掉 Adder Tree 減少 critical path 的長度，可以有較小的 cycle time，在相同 cycle 數下有較高的運算速度，但相較於展開 Loop-1、Loop-2 需要較多暫存器儲存 partial sum。而其面臨的問題是 FC 沒有 Loop-3，會降低 PE Utilization。

4. Unroll Loop-4:

Unroll Loop-4 的好處在於有 input pixel 的 spatial reuse，可以降低 input pixel bandwidth，減少 input pixel 的讀取次數，也拿掉 Adder Tree，有較小的 cycle time，但需要較多暫存器儲存 partial sum。而其面臨的問題是 DWC 沒有 Loop-4，會降低 PE Utilization。

分析與設計:

Unroll Loop-1 的 kernel 大小問題無法解決，Unroll Loop-2 可以提高平行度，但沒有做 data reuse 也沒有降低 output bandwidth，不適合做為主要的 Unroll Loop 方式，Unroll Loop-3 與 Unroll Loop-4，有 data reuse 可以降低 SRAM bandwidth，但無法通用於四種卷積運算。只展開一個特定的 Loop，勢必會使得某一種卷積運算的 PE Utilization 不佳，而我們發現 Unroll Loop-3 和 Unroll Loop-4 具有相同的硬體架構與 data bandwidth 要求，若是可以設計數據流，使加速器可以根據不同卷積運算選擇 Unroll Loop-3 或 Loop-4，

則可以保留 Unroll Loop-3 weight reuse 與 Unroll Loop-4 input pixel reuse 的優點，又可以維持每一種卷積運算的 PE Utilization。

因此我們決定在做一般卷積運算時做 Unroll Loop-3，而 FC 沒有 Loop-3，所以在計算 FC 時，我們設計數據流使 FC Unroll Loop-4，稱此展開方法為 Unroll Loop-3or4。我們設計的數據流如圖 4，而 SRAM 儲存資料的方式如圖 5。在計算一般卷積運算時，我們將 input feature map 存在 bandwidth 較大的 SRAM，weight 則存在 bandwidth 較小的 SRAM，在同個 cycle 內可以讀取一筆 weight 和多筆 input pixel，做到 Unroll Loop-3。在下一層需要計算 FC 時，我們將 output pixel 存入 bandwidth 較小的 SRAM，weight 則存在 bandwidth 較大的 SRAM，如此我們就可以在計算 FC 時 Unroll Loop-4。這樣的數據流讓我們在選擇 Unroll Loop-3 或 Unroll Loop-4 時，不需要兩個 bandwidth 較大的 SRAM，來提供 Unroll Loop-3 時所需的大量 input pixel 和 Unroll Loop-4 時所需的大量 weight，而可以用一個 bandwidth 較小的 SRAM 來做 input pixel 或 weight 的 spatial reuse，降低資料的讀取量，減少功耗，並維持高 PE Utilization。

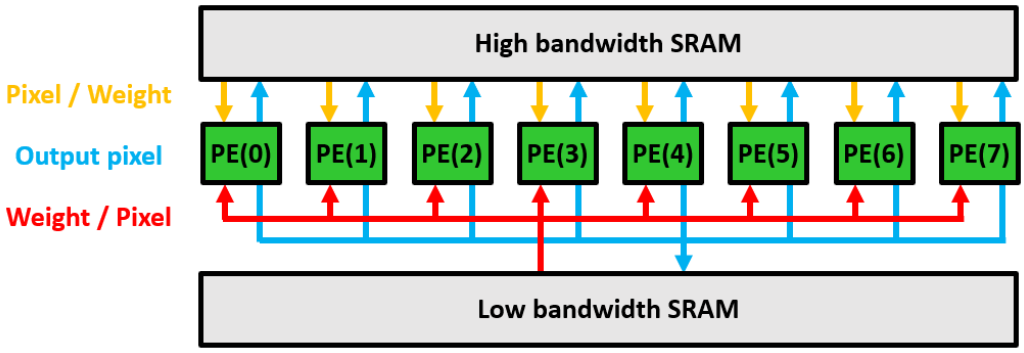

圖 4. 數據流設計

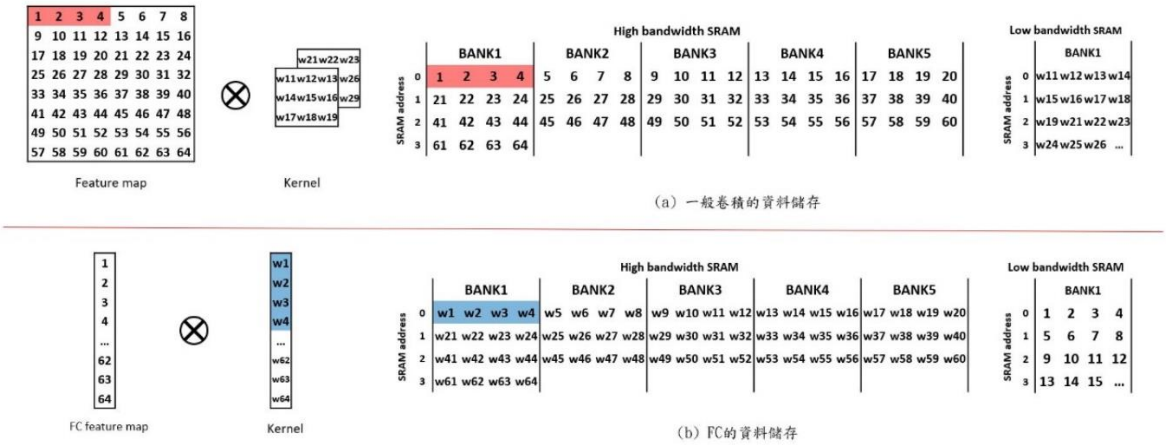

圖 5. SRAM 資料儲存方式

2.1.3. Loop Interchange 設計

Loop Interchange 會影響 data 的 temporal reuse 與 partial sum 的讀寫，也就是影響 SRAM 讀寫量，進而影響功耗。以 SC 為例，每一個 output pixel 是經過 Loop-1, Loop-2 的運算結果，若是我們先計算 Loop-1 和 Loop-2，就可以一次計算完一組 output pixel，不須將 partial sum 存回 SRAM，節省 partial sum 的讀寫。若是我們先計算 Loop-3，可以利用 weight 的 temporal reuse，減少 weight 的讀取。若是我們先計算 Loop-4，可以利用 input pixel 的 temporal reuse，減少 input pixel 的讀取。

我們帶入實際卷積運算做計算，根據 2.1.2 設計的 Unroll Loop-3or4 架構，分析不同 Loop Interchange 對 SRAM 讀寫量的影響。我們得到的結論為，先計算 Loop-1 和 Loop-2 會有較少 SRAM 讀寫量，主要是因為 partial sum 為 20 bits，weight 與 pixel 為 8 bits，partial sum 的讀寫會有較高的代價，且需要額外的 SRAM 來儲存 partial sum，增加電路面積，也有較高的讀寫功耗，因此我們選擇先計算 loop-1 和 loop-2。

再來考慮 Loop-1 和 Loop-2 之間的計算順序，因為主要的運算為 Unroll Loop-3，而 Unroll Loop-3 是展開同一層的 input channel，Loop-1 的計算也是在 kernel 內的同一層 channel，在 Unroll Loop-3 的同時，若最先計算 Loop-1，會有大量重複的 input pixel 在不同的 PE 內運算。如圖 6，3 × 3的 kernel 對8 × 8的 input feature map 做卷積，三個顏色的方框表示做 kernel 內由左到右的運算時會用到的 input pixel，方框內有大量重複的部分，若用 shift register 來暫存 input pixel，可以讓 input pixel 在不同 PE 間有 temporal reuse，每個 cycle 只需 shift 一位，再用控制訊號處理 padding 時輸入為 0，可以減少 input pixel 的讀取次數，若為3 × 3的卷積，可以降到原本的 1/3，以此類推。

而 Loop-3 和 Loop-4 之間的計算順序，先計算 Loop-3 可以完整計算完一個 kernel 再換下一個 kernel，SRAM 的大小只需要夠儲存一個 kernel 就好，若先計算 Loop-4 就需要儲存一層卷積運算中所有的 kernel，為了降低 SRAM 大小我們決定先計算 Loop-3。故我們 Loop Interchange 設計為，先計算 Loop-1、Loop-2、Loop-3，最後計算 Loop-4。

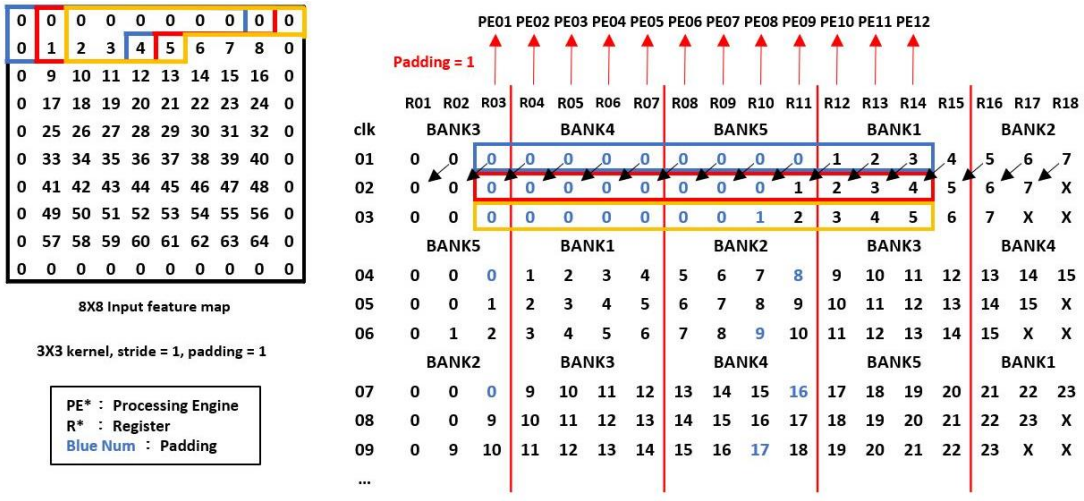

圖 6. SRAM 到 PE 的 input pixel 數據流

2.2 硬體架構設計

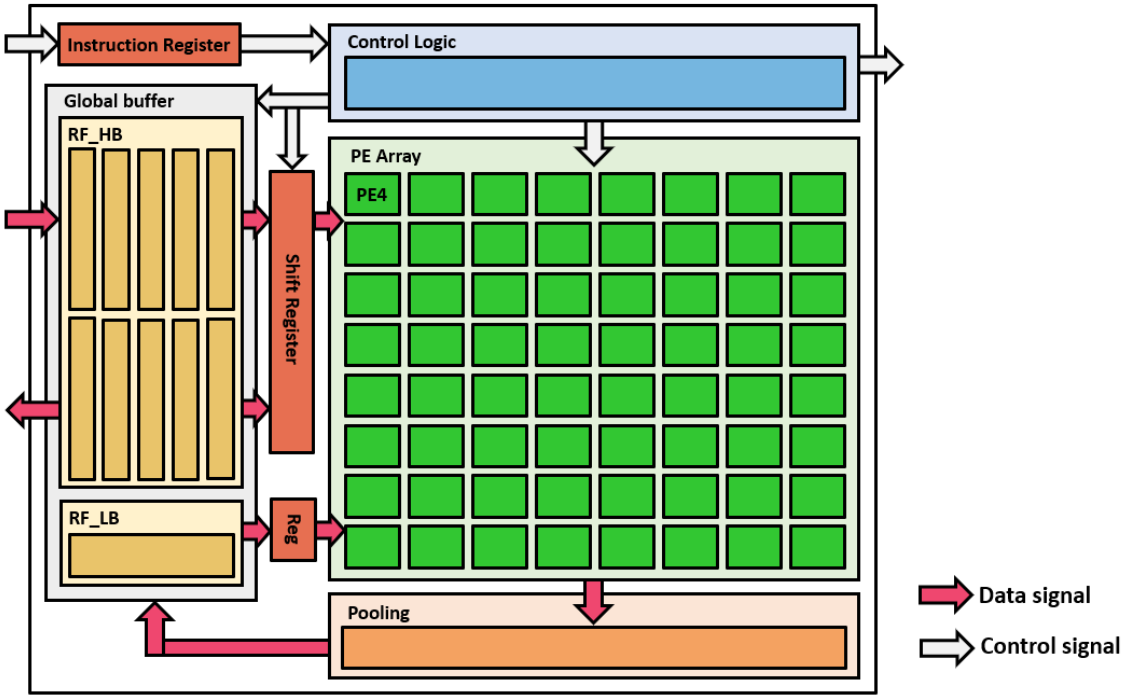

圖 7. 硬體架構圖

2.2.1. 各模塊功能

a. Instruction Register : 暫存器，儲存外部輸入 Instruction 指令
b. Control Logic : 根據 Instruction，產生 RF 需要的 address 與 PE 的控制訊號
c. Global Buffer : 為 Register File 儲存 pixel, weight 資料
d. PE Array : 256 個運算單元(PE)做乘累加運算，並將結果做 ReLU 與 quantization
e. Pooling : 累加與除法器，支援不同大小池化層做 Average pooling

2.2.2. 各模塊實作

a. Instruction Register:

34bits 的暫存器。我們設計的 Instruction 為 34bits。

1. opcode : 卷積運算類型(SC、DWC、FC、SC + POOL、END)
2. Nixy : input feature map 的 x 與 y 方向大小，支援 1 ~ 128
3. Nif : input feature map 的 channel 數，支援 1~1024
4. Nof : output feature map 的 channel 數，支援 1~1024
5. Nkx : kerner 的 x、y 方向大小，支援 1~7
6. S : 步長，支援 1~2

表 2. Instruction 設計

3	7	10	10	3	1
opcode	Nixy	Nif	Nof	Nkx	S

b. PE Array:

運算單元陣列，做乘累加的運算。我們參考其他加速器架構 [3]，設定 input pixel 和 weight 的精確度為 8bits，而 partial sum 的精確度為 20bits。PE Array 中 PE 個數會影響運算效率，PE 數越多運算越快，但相對面積與 SRAM 的 bandwidth 越大，必須在運算速度與面積間取得平衡。

1. PE 數量:

我們希望運算數量級在 10 的 8 次方到 9 次方的 CNN，可以在 10 毫秒左右完成運算，在 clock frequency = 200MHz 的條件下，每個 cycle 大約需要 250 MAC 的計算量，考慮到常見的演算法與 dataset 架構中，feature map 的大小會是8×8或7×7的倍數，為了提高 PE Utilization，我們選擇 Unroll Loop-3or4 8×8的大小，犧牲一點 feature map 大小為7×7倍數時的 PE Utilization。為了達到每個 cycle 所需的計算量，我們選擇 Unroll Loop-2 四個 channel 來增加平行度，共$8\times8\times4=256$個 PE。

2. PE Array 架構:

每個 PE 的輸入分別來自於 Reg 與 Shift Register，Reg 儲存 low bandwidth SRAM 輸出的資料，Shift Register 儲存 high bandwidth SRAM 輸出的資料。Reg 的資料會有 spatial reuse，在一個 cycle 內給多個 PE 做計算，shift register 的資料會有 temporal reuse，在多個 cycle 給不同的 PE 做計算。

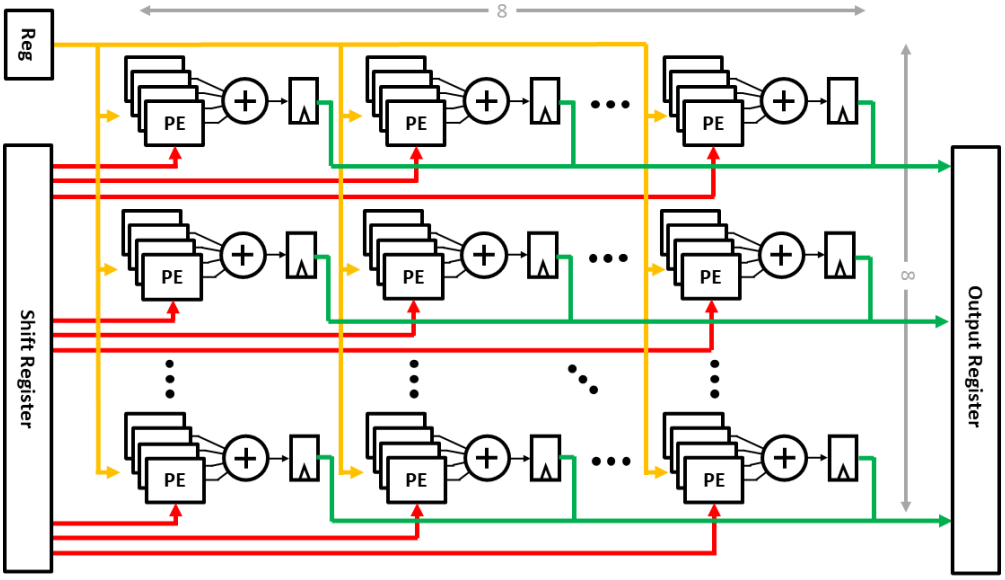

圖 8. PE Array 架構

c. Global Buffer:

1. 儲存量計算

根據我們加速器的設計，我們的 SRAM 最小需要儲存兩個 feature map 與一個 kernel 的資料。表 3 為我們訂定在不同 Nixy 大小下，channel 數限制，計算出需要的總儲存量為 $81.92 + 8.192 = 90.112(KB)$。

表 3. feature map 大小限制

Nixy	Channel 限制	Max Size (KB)
65-128[3]	≤ 3	49.152
33-64	≤ 8	32.768
17-32	≤ 32	32.768
9-16	≤ 128	32.768
7-8	≤ 512	32.768
1-6	≤ 1024	36.864

2. bandwidth 計算

根據 2.1 數據流架構設計，我們設計分成兩個 SRAM，分別為 high bandwidth SRAM (SRAM_HB)與 low bandwidth SRAM (SRAM_LB)。有 256 個 PE，每個 cycle SRAM_HB 至少需要提供 256 筆資料，SRAM_LB 至少需要提供 4 筆資料來維持 PE 運作，又因為我們 shift register 的設計需要增加一點 bandwidth 來減少資料讀取的頻率，因此我們設定 SRAM_HB 的 bandwidth 為10×256bits，SRAM_LB 的 bandwidth 為256bits。

3. Global Buffer 設計

(1) 一開始的設計:

SRAM_HB：二十個 banks，每一個 bank 為 single port SRAM 256×256bits

SRAM_LB：一個 bank，為 single port SRAM 256×256bits

SRAM_HB 需要二十個 banks 是因為需要做 Ping-Pong，同時讀取 input pixel 資料與寫入 output pixel 以維持 PE Utilization，而 2 port SRAM 面積代價太大。但我們遇到的問題是，SRAM 的每個 word 最多只能 144bits，我們需要分成很多 SRAM 才可以有我們需要的 bandwidth，而每個 SRAM 的容量過小，導致面積效益很差。

(2) 最終的設計:

RF_HB：十個 banks，每一個 bank 為 1R1W port Register File 256×256bits

RF_LB：一個 bank，為 1R1W port Register File 256×256bits

在較小的儲存量下，Register File 有比較好的面積效益，且可以在有較小的面積代價下，分成一讀一寫的 port，使控制訊號變得較為簡單。

[3] 限第一層

3. Experimental Results

3.1 數據流比較

3.1.1. Loop Unroll:

為了驗證我們數據流中 Unroll Loop-3or4 設計的通用性，我們在不同類型卷積運算、不同輸入特徵圖大小下，比較 PE 的使用效率。如圖 9，我們將 Unroll Loop-3or4 的設計(This work)與現今常見的 Loop Unroll 方式 [4][4]，在 SC、DWC、PWC、FC 運算且考慮輸入特徵圖為 7 或 8 的倍數時，PE 的使用效率。由結果可以看出，我們的設計在所有類型的卷積運算都有高 PE Utilization，符合我們設計通用性數據流的目標。

1. [Type-(A)] Unroll Loop-1, Loop-2, Loop-4 [5]
2. [Type-(B)] Unroll Loop-2, Loop-4 [6]
3. [Type-(C)] Unroll Loop-1, Loop-3 [7]
4. [Type-(D)] Unroll Loop-3, Loop-4 [4]

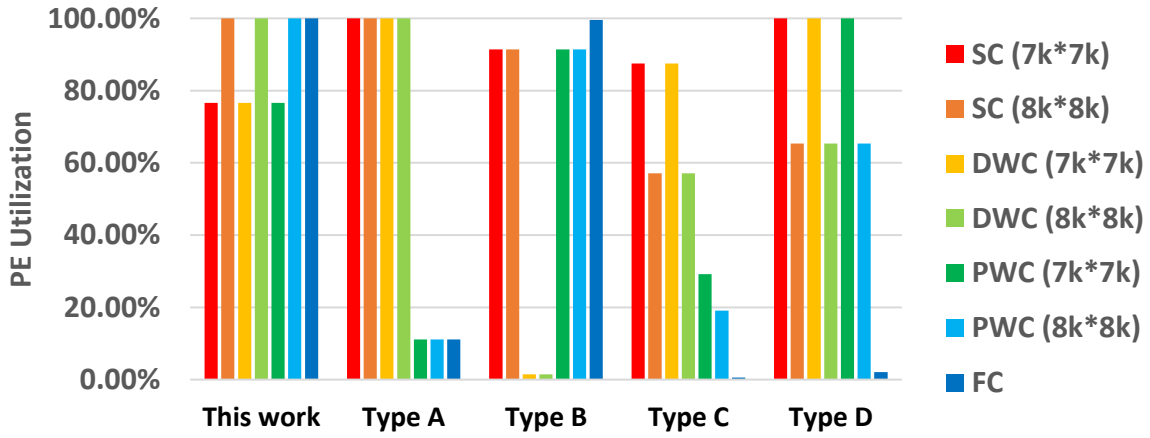

圖 9. PE Utilization 比較

3.1.2. Loop Interchange:

為了驗證我們數據流中 Loop Interchange 的設計是否有效降低 SRAM 讀寫量，我們帶入實際卷積運算，計算各類型資料的讀寫量，並與常見數據流做比較。如圖 10，我們將我們的設計與 Output Stationary (OS)、Weight Stationary (WS)、Input Stationary (IS) 比較，在做 32 層 16×16 input feature map 和 32 個 32 層 3×3 kernel 的 SC 時，各種資料的 SRAM 讀寫量。由結果可以看出，先計算 Loop-1、Loop-2 的 SRAM 總讀寫量是其他數據流的 1/5，而我們 shift register 的設計又將總讀寫量再降為原本的 1/3。我們的設計可以減少 SRAM 讀寫量至其他數據流的 1/15 到 1/3，符合我們降低 SRAM 的讀寫量的目標。

[4] PE 展開個數參考各篇論文

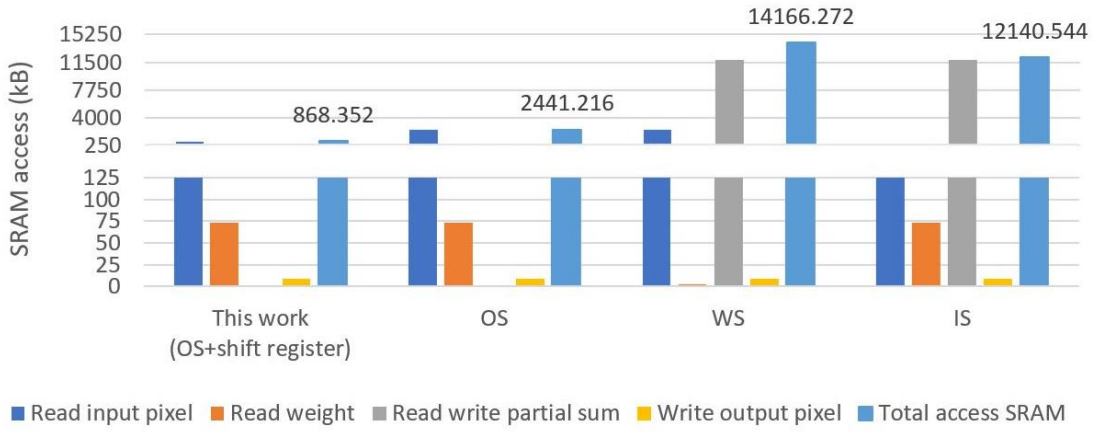

圖 10. SRAM 讀寫量比較

3.2 硬體架構比較

表 4 為我們利用 TSMC 40nm 製成、TT corner、0.9V、25℃電路合成結果[5]，並與其他加速器架構做比較，比較面積效率、能源效率，驗證我們加速器的性能。

表 4. 加速器比較

	This work	Eyeriss [8]	Eyeriss v2 [3]
Technology	40nm	65nm	65nm
Gate Count (NAND-2)	1553.4k gate	1176k gate	2695k gate
On-Chip SRAM	90.1 KB	108 KB	246 KB
Number of PEs	256	168	384
Clock Rate	200 MHz	200 MHz	200 MHz
Bit Precision	8b	16b	8b
Peak Throughput	51.2 GMACS	33.6 GMACS	76.8 GMACS
Area efficiency	32960 (MACS/gate)	28571.4 (MACS/gate)	28497.2 (MACS/gate)
Power	134.2 mW	278 mW (AlexNet)	584 mW (AlexNet)
Power efficiency	381.55 (GMACS/W)	120.9 (GMACS/W)	131.5 (GMACS/W)

[5] 我們加速器仍有功能尚未實現，數據為暫時的結果。

圖 11 為我們電路各個模塊的面積與功耗佔比。可以看出 Global Buffer 的面積佔總面積的 69%，且功耗佔總功耗的 45%，顯示我們利用增加 2%面積的 Shift Register，來降低 memory access 次數是代價小、且可以有效降低功耗的。

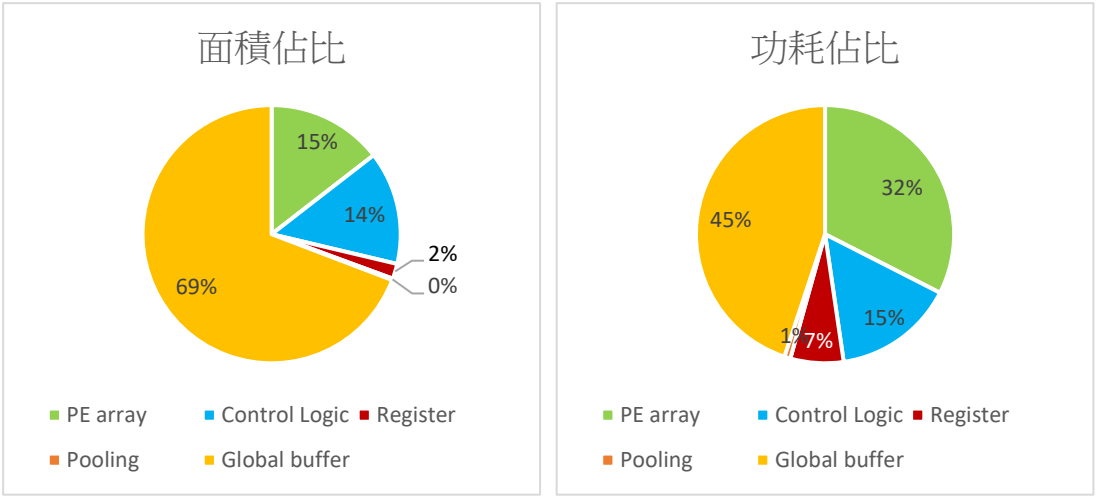

圖 11. 各模組的面積與功耗佔比

4. Conclusion

本專題引入四循環卷積運算分析技巧 [4]，並將數據流的設計與硬體架構設計做結合，提出通用於 Standard convolution (SC), Depthwise convolution (DWC), Pointwise convolution (PWC)和 Fully connect (FC)運算的數據流與硬體架構，並驗證在以上不同類型卷積運算、不同卷積核大小下，若輸入特徵圖為8×8的倍數，有著 100%的 PE 使用效率，若輸入特徵圖為7×7的倍數也有 76.5%的 PE 使用效率，相較於其他數據流設計，有著最高的通用性。並在此硬體架構下，利用數據流與 Shift Register 的設計，有效的增加 data reuse，使 SRAM 的讀寫量為其他數據流的 1/3 至 1/15。最後合成出我們設計的加速器，達到面積效率 32960 (MACS/gate)與能源效率 381.55 (GMACS/W)，並通用於不同類型的卷積運算。

5. Reference

[1] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *arXiv preprint arXiv:1801.04381v4,* 2019.

[2] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861,* 2017.

[3] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze, "Eyeriss v2: A Flexible

Accelerator for Emerging Deep Neural Networks on Mobile Devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems,* vol. 9, no. 2, p. 292 – 308, 2019.

[4] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo, "Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 26, no. 7, pp. 1354-1367, 2018.

[5] K. Guo et al., "Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Des. Integr. Circuits,* 2018.

[6] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA),* 2015.

[7] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *Proc. ACM/IEEE Int. Symp. Comput. Archit. (ISCA),* p. 367–379, 2016.

[8] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, Vivienne Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits,* 2017.

6. Feedback

感謝鄭桂忠教授讓我們有機會實作深度神經網絡加速器。在上學期專題剛開始進行時，我們從一開始對深度學習茫然不解，透過開放課程以及和禹樵學長討論，逐步學習深度學習的基礎知識，再到後來我們開始閱讀各種神經網路演算法的論文，深入的了解卷積神經網路的發展趨勢，以及各個演算法的架構，在這個過程中，我們從演算法的角度出發，去學習別人如何去發想一個新的問題，以及提出新的架構和新的解決方案的思考方式。

在學習完足夠的演算法架構之後，我們開始以硬體的角度出發，思考演算法在硬體加速器上的可行性，以及如何在演算的架構下去優化我們的加速器性能。從軟體轉變成硬體的過程其實也是一大挑戰，因為這需要更加深入了解每個卷積運算的步驟，以及資料如何在儲存單元和運算單元傳遞，從中去找尋將卷積運算平行化的方法，並且嘗試在各種平行化的方法下去最優化我們的性能。在我們開始進行實作之後，我們也遇到了許許多多的問題，很感謝禹樵學長為我們解答了這些疑惑，並且給予我們一些解決方案。