以現場可規劃邏輯閘陣列優化卷積神經網路效能

Optimizing FPGA-Based Convolution Neural Network Performance

陳昭佑

資訊工程系

國立臺南大學

目錄

目錄

深度學習運算平台比較

	CPUs	GPUs	ASICs	FPGAs
耗能	✕	✕	○	○
靈活性	○	△	✕	○
訓練	✕	○	△	✕
成本	○	✕	✕	○
價格	△	✕	✕	△
推斷力 (inference)	✕	△	✕	○

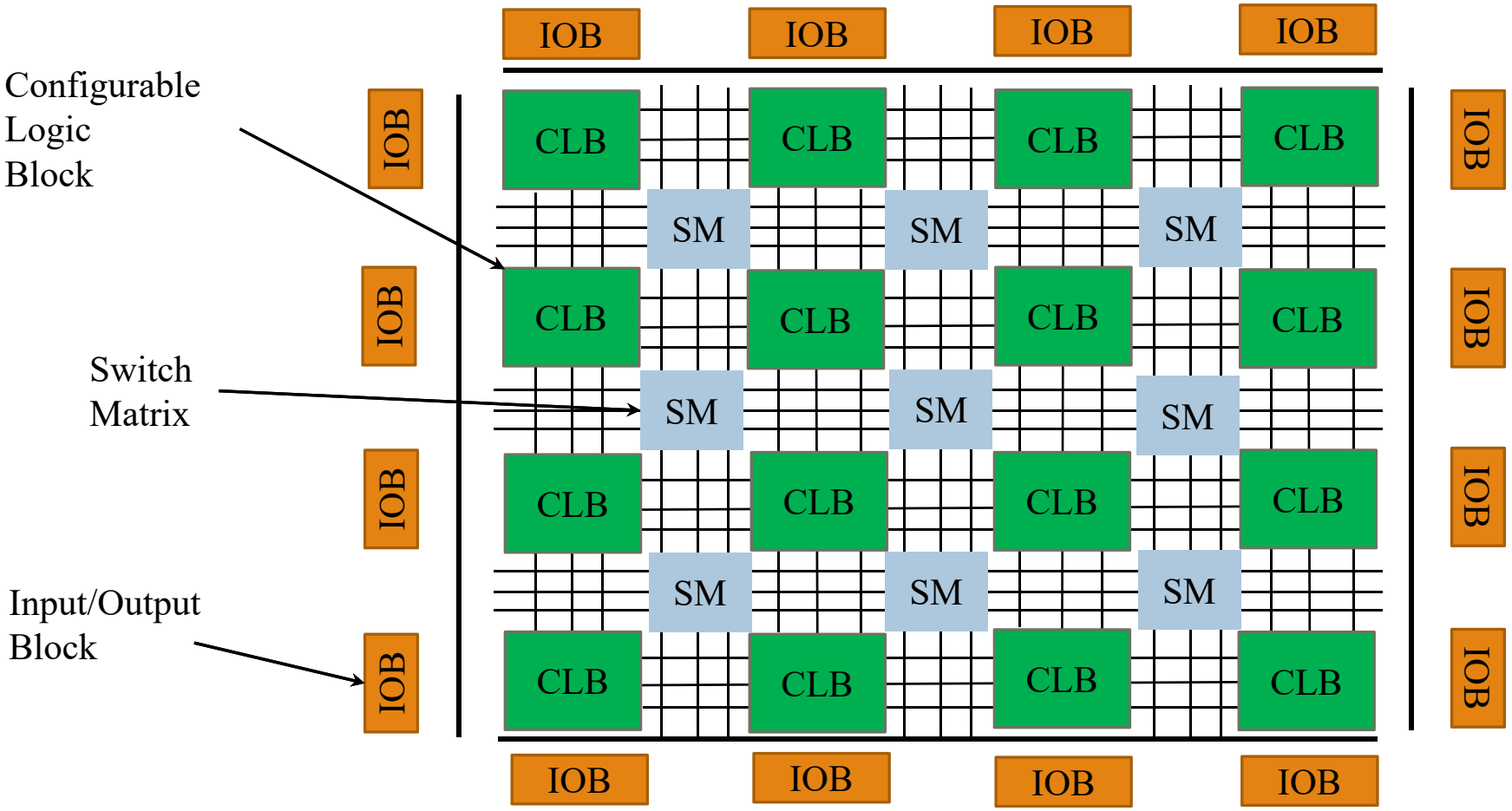
推斷力 ： 把從訓練中學習到的結果應用在工作中　　○:優　△:尚可　✕:差

FPGA的優勢

- 資料類型不限
- 低功耗
- 低成本
- 應用範圍廣

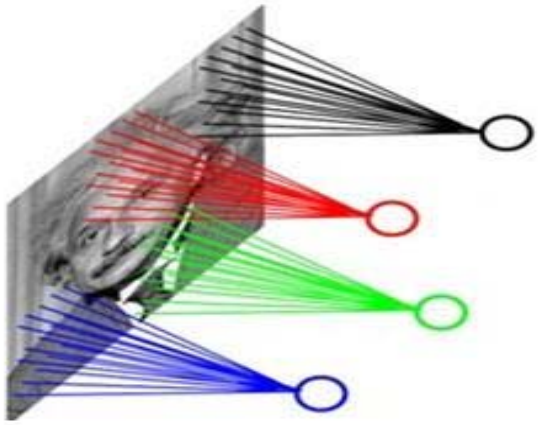
目錄

FPGA架構


Configurable
Logic
Block
Switch
Matrix
Input/Output
Block
IOB
CLB
SM

CNN在圖像識別之技術 -局部感知(Receptive Field)

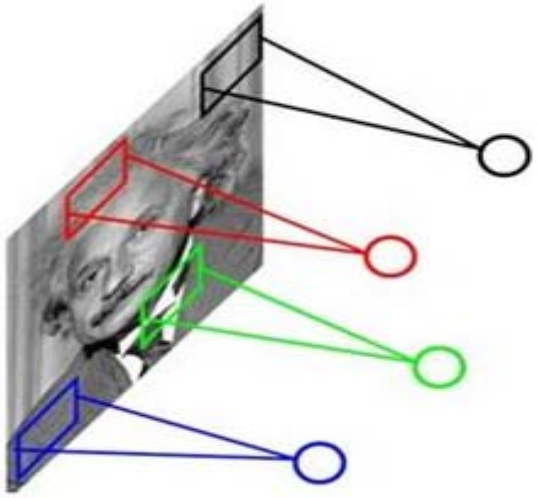
全連接模式(經典神經網路)



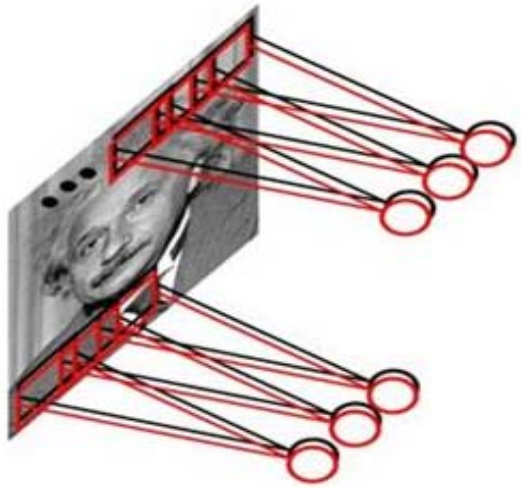
局部連接模式(卷積神經網路)

- 每個神經元只與上一層的部分神經元相連，只感知局部，而不是整幅圖像。
- 省去不必要的計算進而節省時間。

CNN在圖像識別之技術 –參數共享(Shared Weights And Biases)



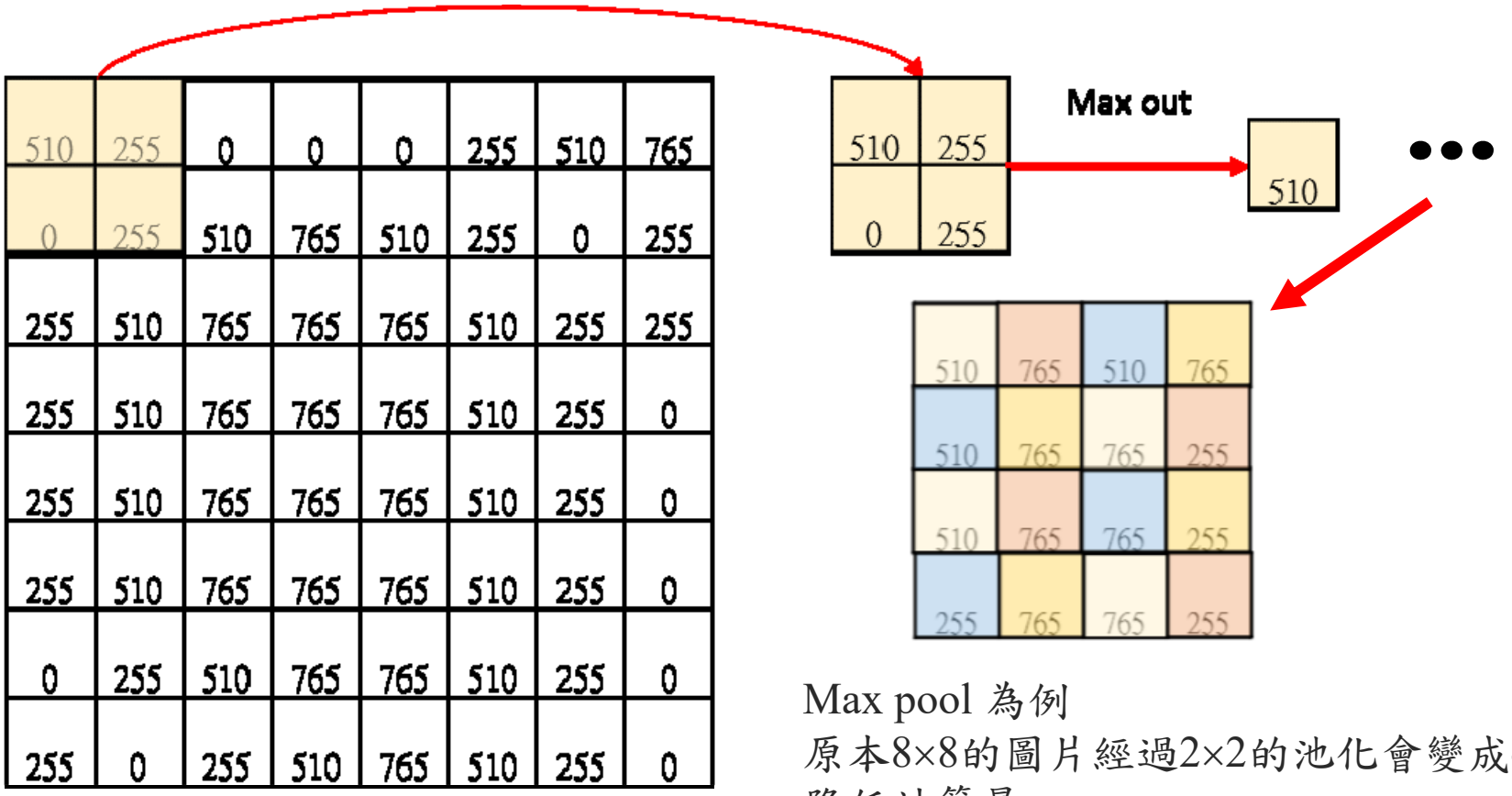
參數獨立



參數共享

- 在提取一個個局部信息當中這一部分學到的特徵也可以用到另一部分上。所以對圖像上的所有位置，都能使用同樣的學習特徵。
- 減少參數量。

CNN在圖像識別之技術 –池化(Polling)



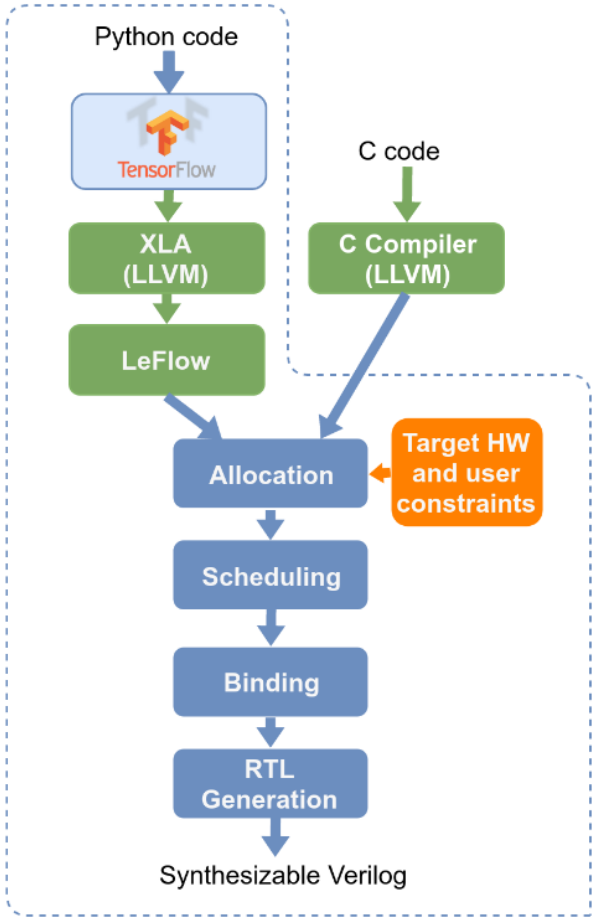
510	255	0	0	0	255	510	765
0	255	510	765	510	255	0	255
255	510	765	765	765	510	255	255
255	510	765	765	765	510	255	0
255	510	765	765	765	510	255	0
255	510	765	765	765	510	255	0
0	255	510	765	765	510	255	0
255	0	255	510	765	510	255	0

Max pool 為例
原本8×8的圖片經過2×2的池化會變成4×4
降低計算量

CNN缺點

- 準確度高，但複雜度高，運算量大。
- 軟體實現的效率比硬體實現差

FPGA的CNN設計帶來的挑戰一



- TensorFlow：深度學習框架，支援各式不同深度學習演算法。
- 將FPGA的CNN設計做集成，將帶來一系列TensorFlow中的**集成開銷**(Integration Overheads)。

FPGA的CNN設計帶來的挑戰二

- CNN的複雜架構，導致以FPGA硬體實現時，可能會增加計算複雜度。

- 目前存在降低計算複雜度的方法–深度可分離卷積(Depth-wise Separable Convolutions, DSC)

DSC和一般的計算量比較

- 深度卷積(depthwise convolution)：
  輸入資料: $Win \times Hin \times Nch$ ， $k \times k$ 的convolution
  輸出大小: $Wout \times Hout \times Nch$
- 逐點卷積(pointwise convolution)：
  輸入資料: $Win \times Hin \times Nch$，$1 \times 1$ 的convolution共 Nk個，
  輸出大小: $Wout \times Hout \times Nch$

$$\frac{\text{DSC計算量}}{\text{一般卷積計算量}}$$

$$= \frac{Win \times Hin \times Nch \times k \times k + Win \times Hin \times Nch \times Nk}{Win \times Hin \times Nch \times k \times k \times Nk}$$

$$= \frac{1}{Nk} + \frac{1}{k \times k}$$

W :寬
H :高
N :個數
k : kernel map
in : 輸入
out : 輸出

DSC的問題

- CNN在不同層各有各自的輸入和輸出、特徵圖大小和kernel大小。

- 難以用同一種架構實現不同層，需以不同的方式處理不同層以優化效能。

研究目的

1. 使用FPGA設計卷積神經網路提高效率。

2. 使用動態平鋪(Dynamic Tiling)技術設計加速器以優化跨層的硬體效能。

目錄

- 介紹
- 文獻回顧
  - 現場可規劃邏輯閘陣列(FPGA)
  - CNN在圖像識別之技術
  - FPGA的CNN設計所帶來的挑戰
- 研究方法
  - 具彈性與整合性的管線化軟硬體架構
  - 動態平鋪
- 實驗結果
  - 實驗環境
  - 實驗結果
  - 實驗比較
- 結論

挑戰一–降低集成開銷

問題 ：

- 集成開銷→需要花費更多的時間

方法 ：

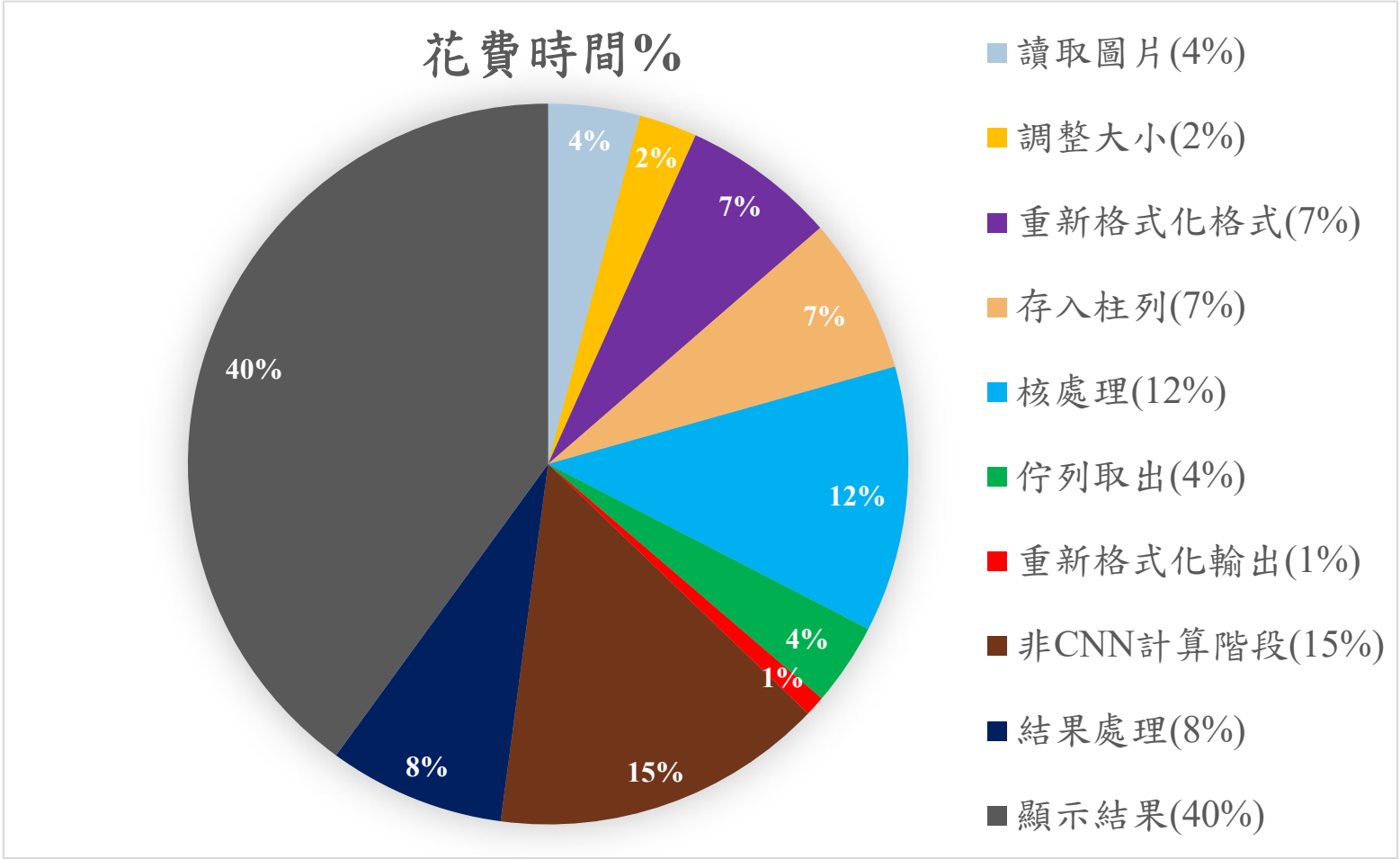
- 具彈性與整合性的管線化(pipelining)軟硬體架構。

具彈性與整合性的管線化軟硬體架構

為了從TensorFlow實行FPGA，需要在重新定義了節點，其餘部分在CPU上處理。步驟如下 ：

1) 讀入CNN的輸入
2) 前處理圖像大小
3) 重置CPU內中的初始數據
4) 將數據從CPU傳輸到FPGA
5) 在FPGA上進行計算
6) 通過PCIe取得結果
7) 重置並將其傳遞給TensorFlow
8) 在CPU上做非CNN的計算階段
9) 處理結果
10) 寫出並顯示結果

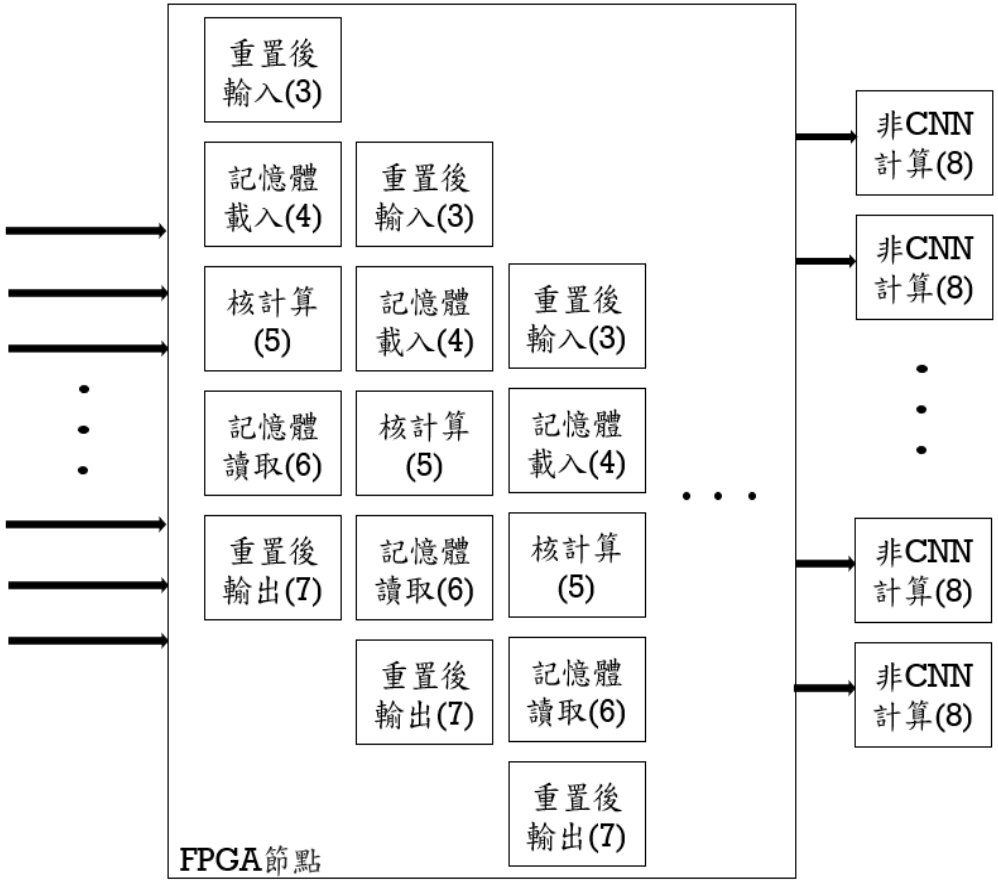
集成步驟花費時間分析


花費時間%
4%
2%
7%
7%
12%
4%
1%
15%
8%
40%
讀取圖片(4%)
調整大小(2%)
重新格式化格式(7%)
存入柱列(7%)
核處理(12%)
佇列取出(4%)
重新格式化輸出(1%)
非CNN計算階段(15%)
結果處理(8%)
顯示結果(40%)

兩階段管線化 (two-level pipelining)

- 第一階段是框架的開銷
  - 步驟包含了1、2、9和10。
- 第二階段為管線化架構中以FPGA計算資料重疊步驟
  - 步驟包含3、4、6和7。

管線化軟體架構流程圖


重置後
輸入(3)
記憶體
載入(4)
重置後
輸入(3)
核計算
(5)
記憶體
載入(4)
重置後
輸入(3)
記憶體
讀取(6)
核計算
(5)
記憶體
載入(4)
重置後
輸出(7)
記憶體
讀取(6)
核計算
(5)
重置後
輸出(7)
記憶體
讀取(6)
重置後
輸出(7)
非CNN
計算(8)
非CNN
計算(8)
非CNN
計算(8)
非CNN
計算(8)
FPGA節點

編譯系統的組成

- 設計空間探索引擎(Design Space Exploration Engine, DSE)

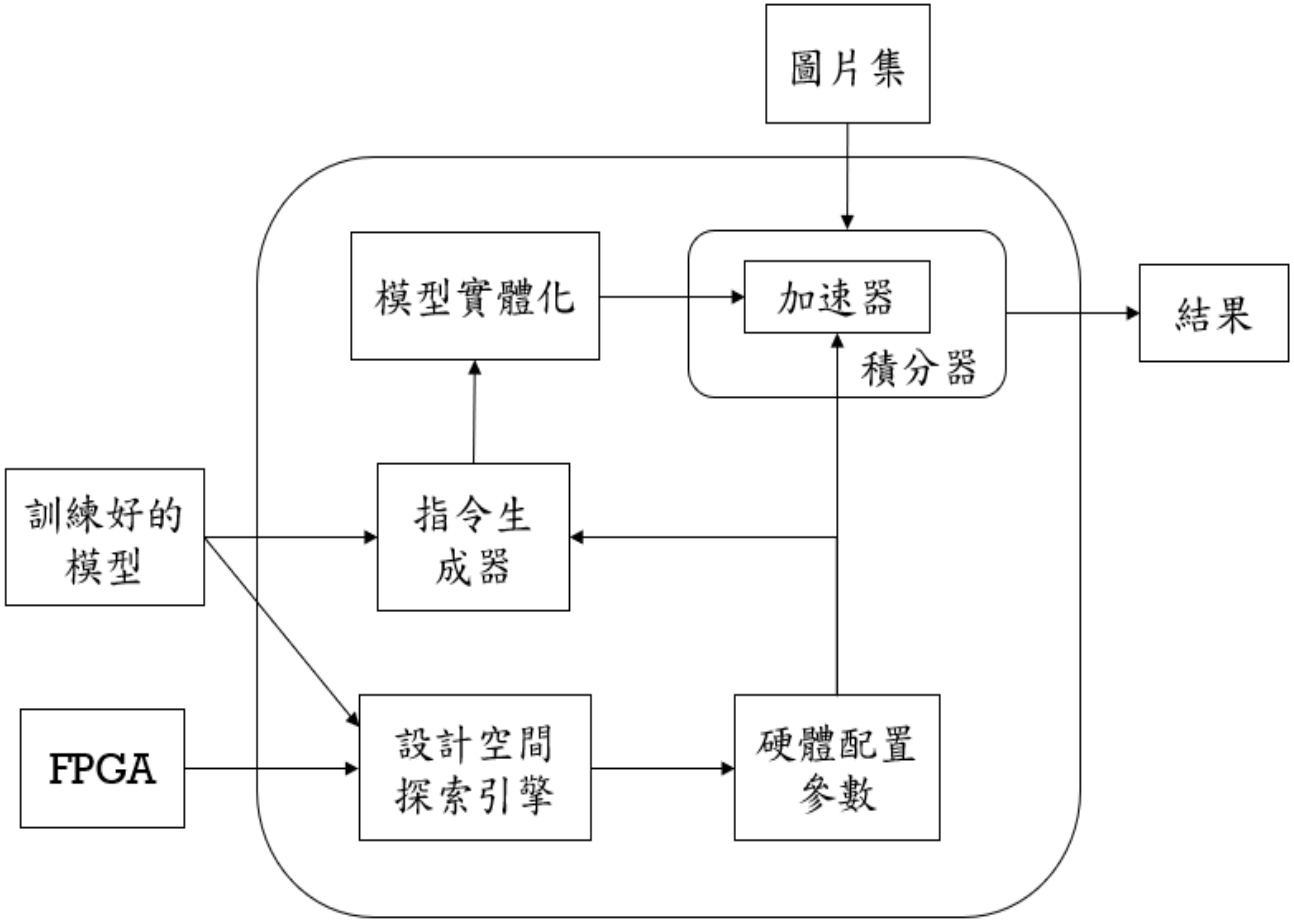
  FPGA會將資源送到設計空間探索引擎中，以搜索最佳的硬體配置參數。

- 指令生成器(Instruction generator, IG)：

  指令生成器將CNN模型和加速器硬體描述文件作為輸入，並為每個CNN層創建指令提高計算效率。

- 積分器(Integrator) ：積分器採用FPGA加速器輸出到框架中，執行點到點處理任務。

系統硬體架構



挑戰二–降低計算複雜度

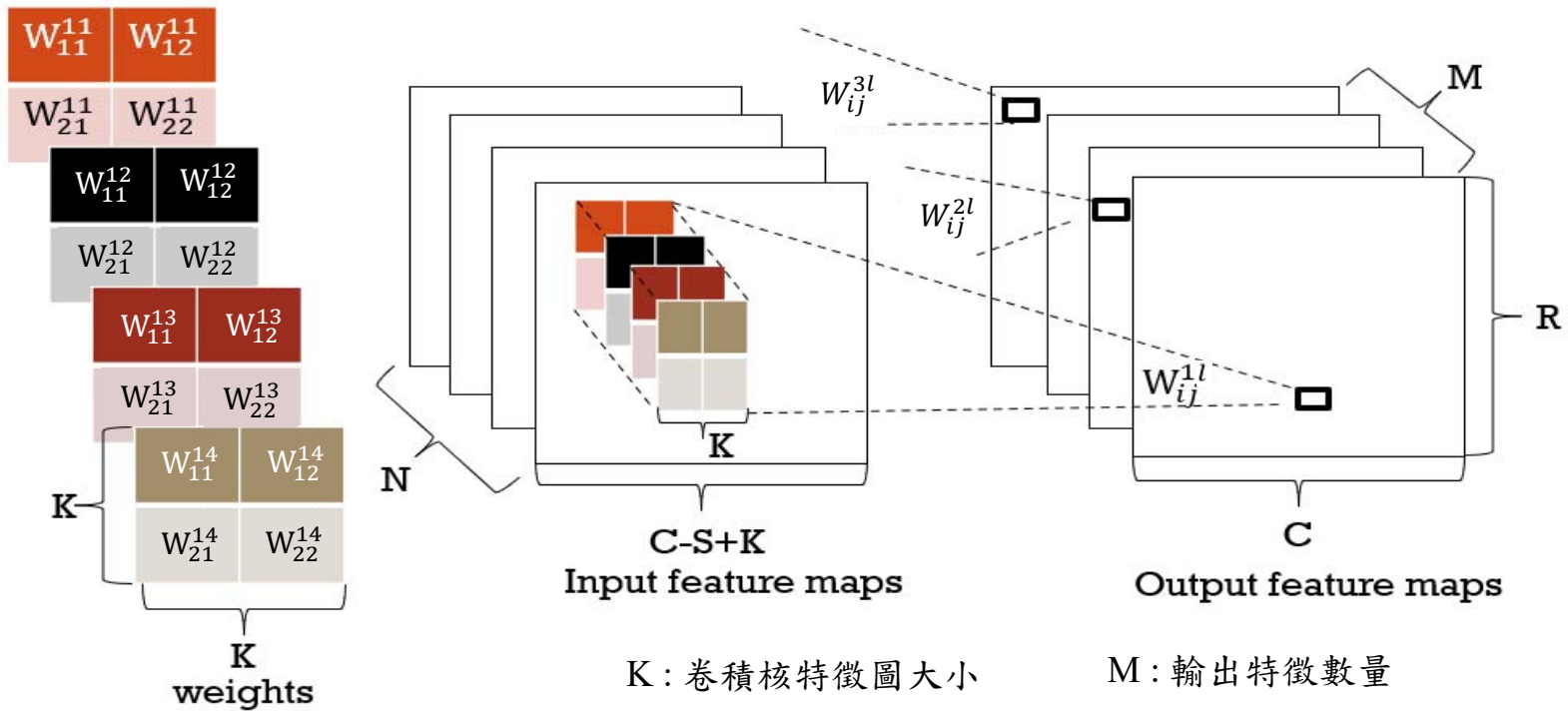
問題：

- CNN的複雜架構，導致以FPGA硬體實現時，可能會增加計算複雜度。

方法 ：

- 使用FPGA加速器卷積運算
- 動態平鋪(Dynamic Tiling)

卷積層的計算



K : 卷積核特徵圖大小

C : 輸出特徵圖的高

S ：滑動間隔

N : 輸入特徵數量

M : 輸出特徵數量

R :輸出特徵圖的寬

W : 權重

l ： 第 l 個卷積後的特徵

卷積層演算法

```
for(row = 0; row < R; row ++) //輸出特徵圖的寬
 for(col = 0; col < C; col ++) //輸出特徵圖的高
  for(to = 0; to < M; to ++) //輸出特徵通道數
   for(ti = 0; ti < N; ti ++) //輸入特徵通道數
    for(i = 0; i < K; i++)   //卷積核特徵圖大小
     for(j = 0; j < K; j++)  //卷積核特徵圖大小
       O[to][row][col] += W[to][ti][i][j] * I[ti][S * row + i][S * col + j];
```

K : 卷積核特徵圖大小

C : 輸出特徵圖的高

S ：滑動間隔

N : 輸入特徵數量

M : 輸出特徵數通道數

R :輸出特徵圖的寬

W : 權重

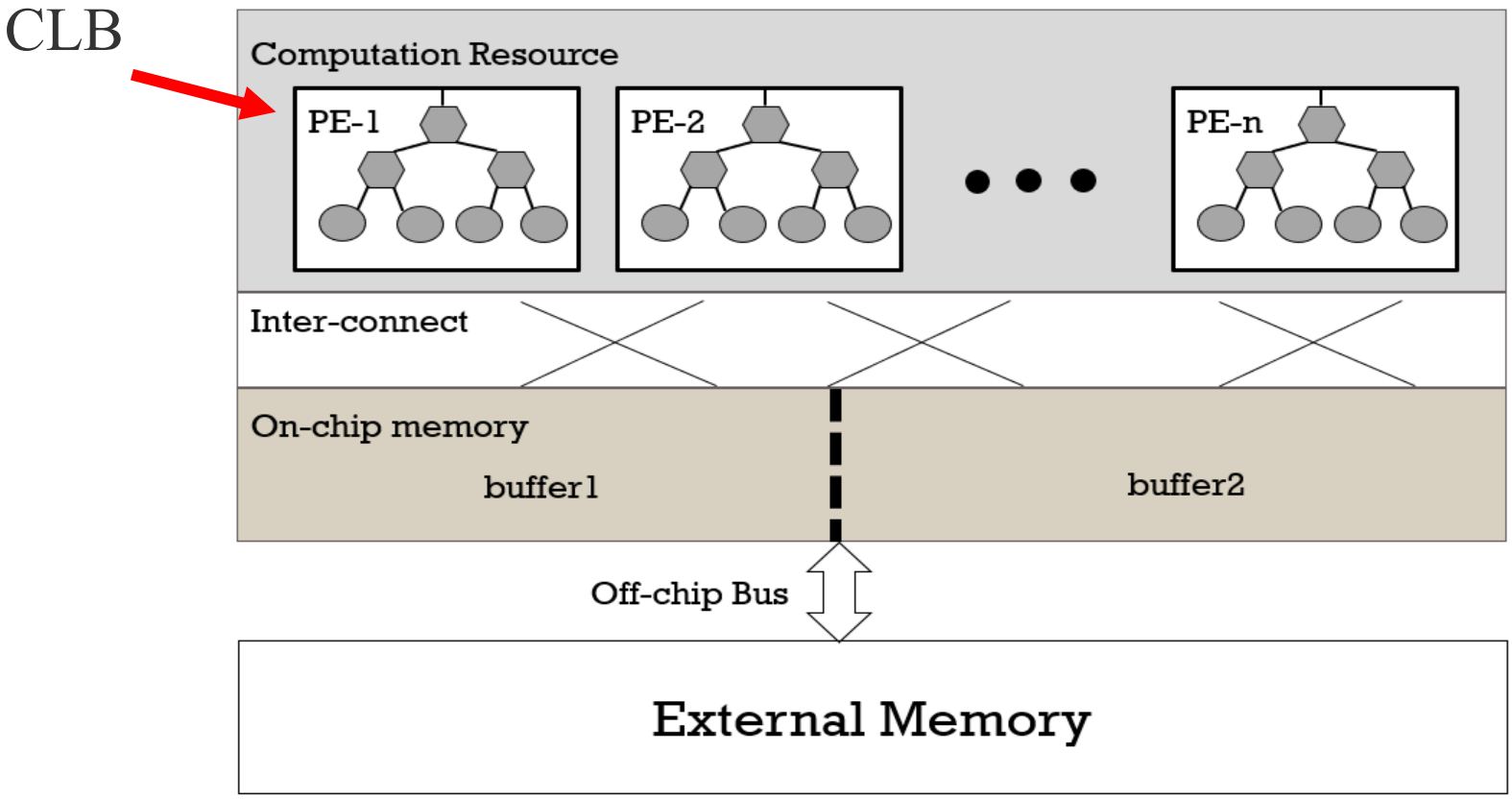
O : 輸出陣列

W : 權重陣列

I ： 輸入陣列

row、col、to、ti、i、j : 變數

使用FPGA加速器卷積運算架構圖



FPGA設計CNN加速器設計

- 迴圈分塊(loop tiling) 改善資料重用和資料並行處理效率。
- PE傳到buffer應仔細分配，這樣能高效處理片上資料。
- PE的資料處理吞吐應該匹配FPGA平臺的外部存儲器頻寬。

動態平鋪演算法

```
for(row = 0 ; row < R; row +=Tr)
 for(col = 0 ; col < C; col +=Tc)
  for(to = 0 ; to < M; to +=Tm)
   for(ti = 0 ; ti < N; ti +=Tn)
   //load I/O feature maps、weights
```

K：卷積核特徵圖大小　M：輸出特徵數通道數

C：輸出特徵圖的高　R：輸出特徵圖的寬

S：滑動間隔　I：輸入陣列

N：輸入特徵數量　W：權重陣列　O：輸出陣列

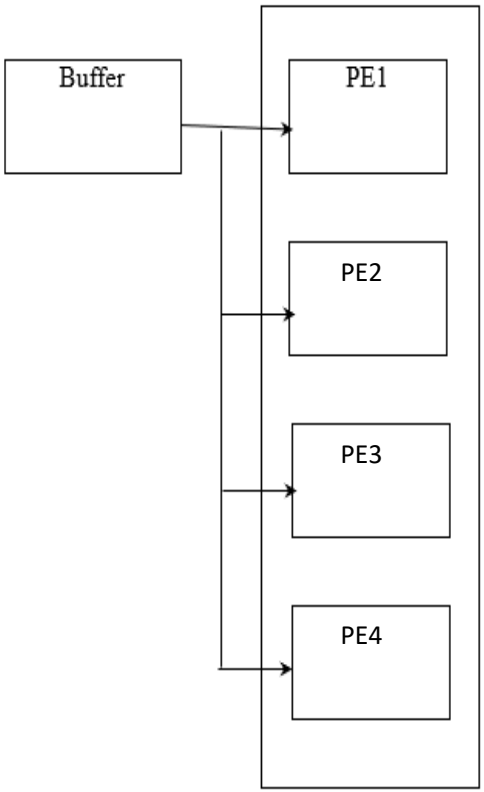
```
for (trr = row ; trr < min (row + Tr, Tr); trr++)
  for (tcc = col ; tcc < min (col + Tc, C); tcc++)
    for (too = to; too < min (too + Tm, M); too++)
      for (tii = ti; tii < min (ti+ Tn, N); tii++)
        for (i = 0; i < K; ti++)
          for (j = 0; j < K; tj++)
            O[too][trr][tcc] += W[too][tii][i][j]*
                                I[tii][S*trr+i] [S*tcc+j]
//store output feature maps
```

平鋪結構

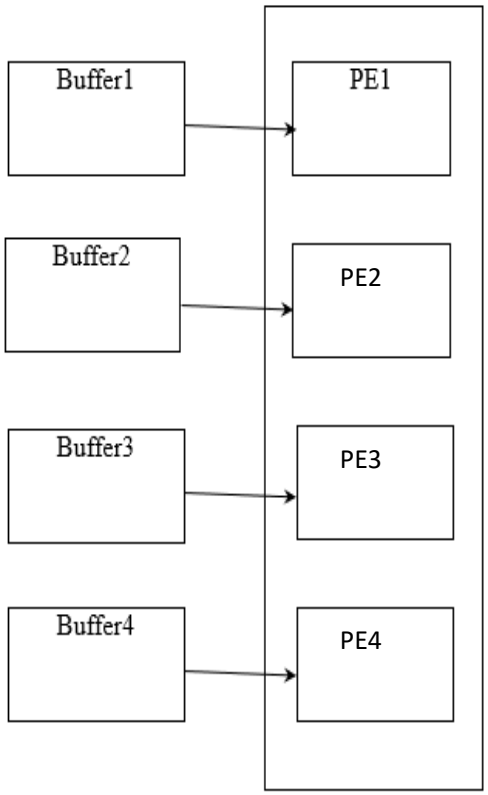
row、*col*、*to*、*trr*、*tcc*、*too*、*tii*、*ti*、*i*、*j*：變數

$0 < T_m \leq M$
$0 < T_n \leq N$
$0 < T_r \leq R$
$0 < T_c \leq C$

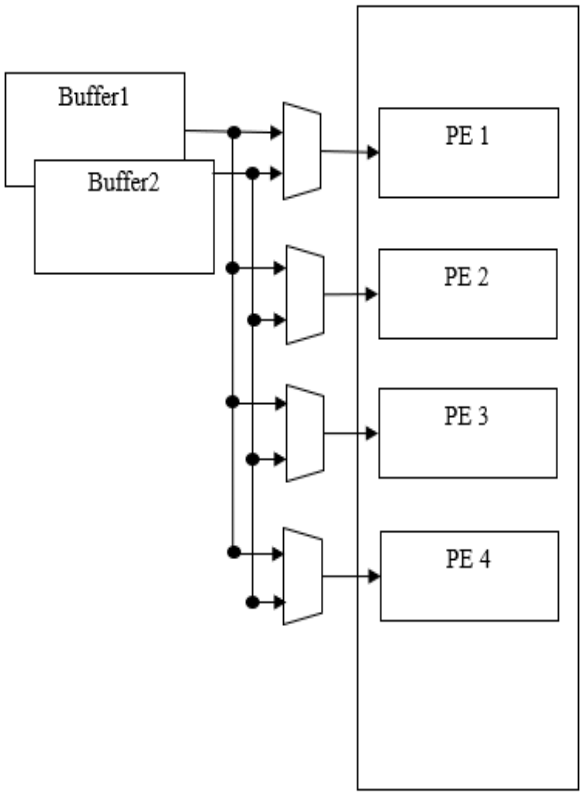
參數的共享關係



不相干



獨立



依賴

動態平鋪參數關係

```
for(row = 0 ; row < R; row +=Tr)
 for(col = 0 ; col < C; col +=Tc)
  for(to = 0 ; to < M; to +=Tm)
   for(ti = 0 ; ti < N; ti +=Tn)
```

N：輸入特徵數量　M：輸出特徵數通道數　$0 < T_m \leq M$

C：輸出特徵圖的高　R：輸出特徵圖的寬　$0 < T_n \leq N$

S：滑動間隔　I：輸入陣列　$0 < T_r \leq R$

K：卷積核特徵圖大小　W：權重陣列　$0 < T_c \leq C$

O：輸出陣列

row、*col*、*to*、*trr*、*tcc*、*too*、*tii*、*ti*、*i*、*j*：變數

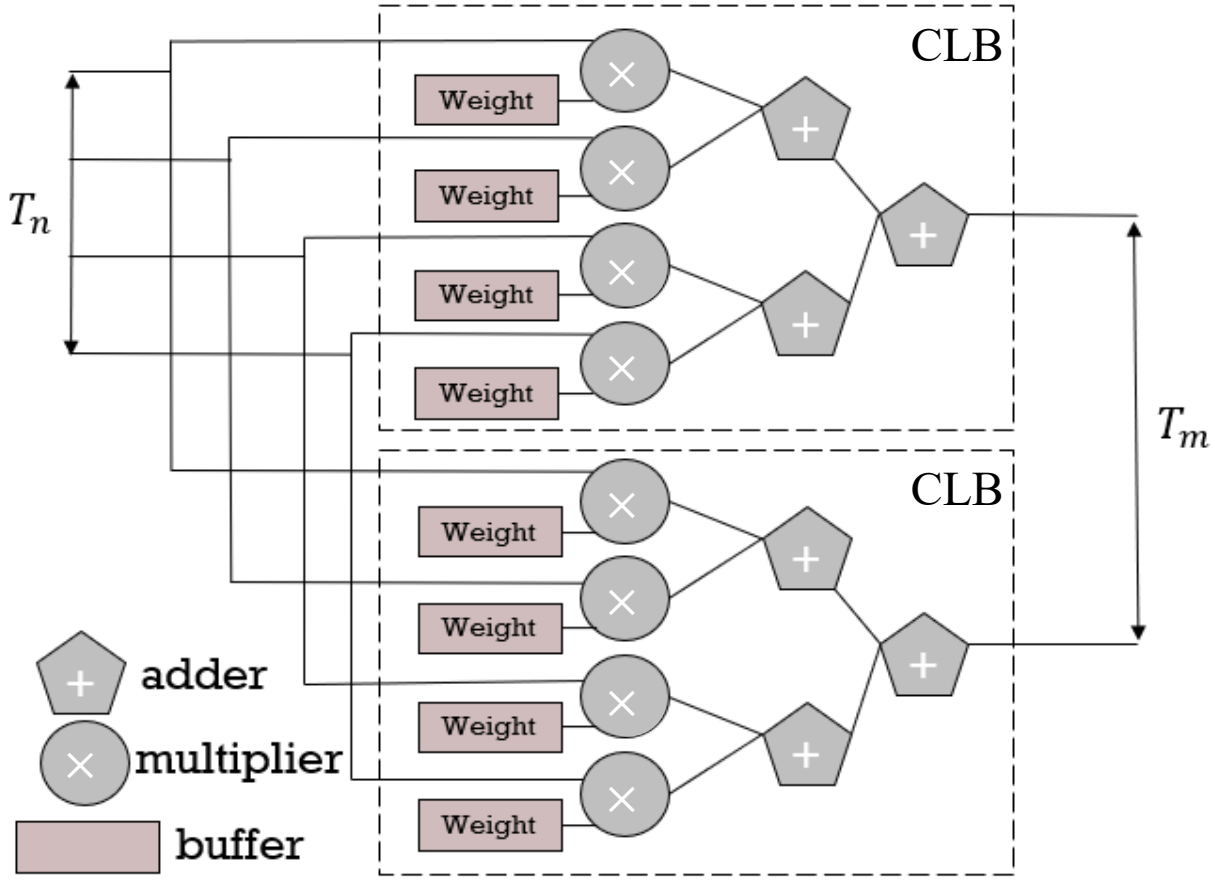
//load I/O feature maps、weights

```
for (trr = row ; trr < min (row + Tr, Tr); trr++)
  for (tcc = col ; tcc < min (col + Tc, C); tcc++)
    for (too = to; too < min (too + Tm, M); too++)
      for (tii = ti; tii < min (ti+ Tn, N); tii++)
        for (i = 0; i < K; ti++)
          for (j = 0; j < K; tj++)
            O[too][trr][tcc]
           += W[too][tii][i][j]*
                 I[tii][S*trr+i] [S*tcc+j]
```

	I	W	O
trr	依賴	不相干	獨立
tcc	依賴	不相干	獨立
too	不相干	獨立	獨立
tii	獨立	獨立	不相干
i	依賴	獨立	不相干
j	依賴	獨立	不相干

//store output feature maps

硬體實現


CLB
Weight
Weight
Weight
Weight
T_n
T_m
CLB
Weight
Weight
Weight
Weight
adder
multiplier
buffer

目錄

- 介紹
- 文獻回顧
  - 現場可規劃邏輯閘陣列(FPGA)
  - CNN在圖像識別之技術
  - FPGA的CNN設計所帶來的挑戰
- 研究方法
  - 具彈性與整合性的管線化軟硬體架構
  - 動態平鋪
- 實驗結果
  - 實驗環境
  - 實驗結果
  - 實驗比較
- 結論

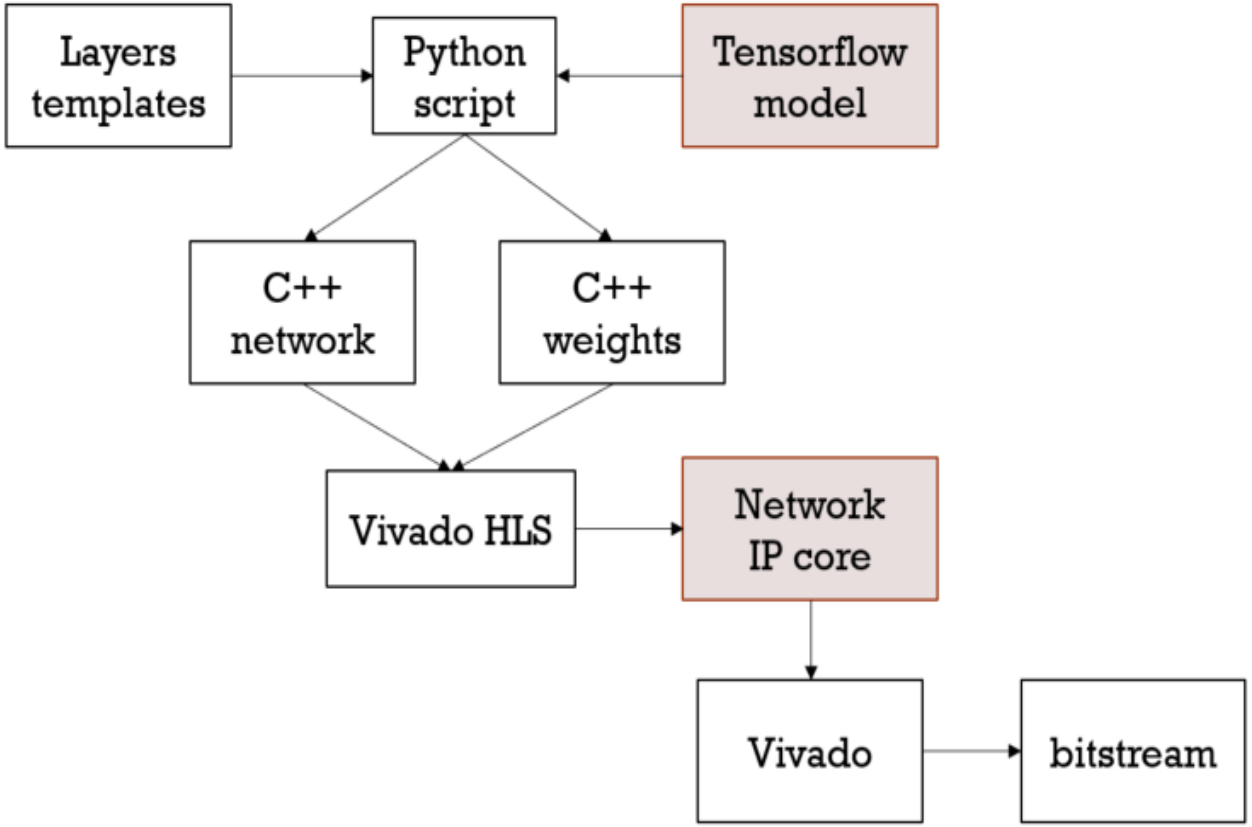
實驗結果

- 平台是ALINX AX7050: Spartan-7 XC7S50
- 測試資料方面是使用MNIST做測試

MNIST (數字手寫圖片集):

Modified National Institute of Standards and Technology Database，一個大型手寫數字數據集，通常用於訓練深度學習圖像系統。

程式運作流程


Layers templates
Python script
Tensorflow model
C++ network
C++ weights
Vivado HLS
Network IP core
Vivado
bitstream

不同平台的比較

Platform	Runtime (*ms*)	Comparison	Power (W)	Comparison
CPU	95.6	1	20	1
GPU	20.3	78.7%	49	-145%
FPGA	25.6	75.2%	15	25%

CPU : Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
GPU : NVIDIA GeForce GTX 1050 Ti
FPGA : ALINX AX7050: Spartan-7 XC7S50

集成開銷 (Integration Overheads)改善

Design	Runtime(*ms*)	Comparison
no pipeline	205.6	1
two pipeline	119.6	41.8%

FPGA加速器設計參考文獻

- **X. Wei, et al.**, "FPGA Acceleration of Deep Convolutional Neural Networks," The 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2017.

- **Y Chen, ed al.**, "Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs," The 54th Annual Design Automation Conference 2017, 2017.

- **J. Zhang, ed al.**, "Improving the Performance of OpenCL-based FPGA Accelerator for Convolutional Neural Network," The 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2017.

FPGA加速器設計比較結果

	X. Wei	Ours	Y. Chen	Ours	J. Zhang	Ours
Model	AlexNet		VGG		ResNet	
Time (*ms*)	4.05	1.03	17.18	22.35	71.71	17.34
Improvement	1	74.6%	1	-30.1%	1	75.8%
Average	40.1%					

目錄

結論

- 我們提出解決深度學習應用於FPGA的兩個主要挑戰:
  - 降低FPGA在深度學習中的集成開銷。
  - 設計FPGA加速器降低計算複雜度度。
- 針對第一個挑戰，提出使用**具彈性與整合性的管線化軟硬體架構**來降低集成開銷。
- 針對第二個挑戰，提出使用**動態平鋪**來降低計算複雜度。
- 實驗結果證明本研究方法能夠優化性能。