Software and Hardware Implementation of Pose Estimation Algorithm Based on 4-bit Quantization

基於 4 位元量化姿態辨識演算法之軟硬體實作

專業領域：系統組
組別： B147
指導教授：呂仁碩
組員姓名：蔡惠芸、吳書磊、張博閔、陳泰融

Abstract

在深度學習的發展中，深度學習模型訓練與推論所需要花費的運算資源以及時間都變得越大量，如何減少這運算資源以及時間是重要的課題。本專題針對姿態辨識演算法 HRNet 的推論部分進行運算資源優化以及計算過程加速。

本研究為達成上述目的，嘗試將推論所需運算量化，並且因為推論的核心計算為矩陣乘法，為了能降低記憶體存取的時間和能源消耗，來縮減計算資源的花費，我們利用在硬體端實作脈動式陣列來做矩陣運算。

最後本專題在軟體方面實作了三種不同的量化版本，其中權衡辨識效能及儲存空間、推論時間等因素後選取 4 位元版本作為硬體實作之規格。而硬體實作上以 Verilog 硬體描述語言進行模擬，主體採用的是脈動式陣列搭配其他控制電路，達成功能正確以及可使用 Verilog 模擬之效果。未來如果有機會能克服硬體空間大小的限制，期許能實際將模型量化之資料放進去脈動式陣列進行運算或是使用 FPGA 板實際演示本專題之硬體電路功能。

Introduction

隨著各式各樣的深度學習研究發展，深度學習開始廣泛的應用在日常生活中。而追求更強大的深度學習應用的同時，深度學習模型層數也變得更深、參數變得更多，使得深度學習網路變得越來越複雜。

越複雜的模型，要訓練(training)以及推論(Inference)所花的運算資源以及時間就會越大量，而如何縮減運算資源的利用以及時間的加速是重要的課題。

在這次研究中，我們選擇了姿態辨識領域的深度學習演算法 HRNet，針對推論這個部份來進行優化。在推論階段的計算工作如下，各個神經網路層把輸入資料乘上參數、將結果傳遞給下一層當成輸入、到達最後一層產生出模型推論的結果。

本專題先透過在軟體端量化模型，縮減運算資源以及加速推論時間。採用的量化模式為 QAT(Quantize Aware Training)，QAT 於訓練過程中即考慮將參數量化，正向傳遞時用量化後之浮點數進行乘加運算，反向傳遞時則針對量化前之浮點數進行權重更新，此模式可使模型更加適應於量化後之數值。

首先介紹針對權重進行量化之方法，由於推論時權重為訓練完成後所得到之定值，因此訓練及推論時正向傳遞(forward propagation)之流程完全相同。

$$r_{max,w} = \max(W), r_{min,w} = \min(W), \text{N: number of bits} \tag{2.1}$$

$$S_w = \frac{r_{max,w} - r_{min,w}}{2^N - 1} \tag{2.2}$$

$$w_{Qi} = round\left(\frac{w_i - r_{min,w}}{S_w}\right) \tag{2.3}$$

$$w_{DQi} = w_{Qi} \cdot S_w + r_{min,w} \tag{2.4}$$

對一個三維卷積層，取其權重最大值及最小值作為量化之區間，並視欲量化之位元數(N)將此區間均分為$2^N - 1$等分，如式(2.2)所示。接著將所有浮點數之權重依式 (2.2)得到之尺度(scale)近似為0至$2^N - 1$之量化整數值，如式(2.3)所示。最後將整數值進行反量化(dequantize)為量化之浮點數以進行正向傳遞運算。

接著介紹對輸入特徵圖進行量化之方法。概念與權重之量化方法相近。

$$r_{max,x} = \text{avg}(\max(X^B)), r_{min,x} = \text{avg}(\min(X^B))$$
$$\text{B: batch size, N: number of bits} \tag{2.5}$$

$$S'_x = \frac{r_{max,x} - r_{min,x}}{2^N - 1} \tag{2.6}$$

$$x_{Qi} = clamp\left(0, 2^N - 1, round\left(\frac{x_i - r_{min,x}}{S'_x}\right)\right) \tag{2.7}$$

$$x_{DQi} = x_{Qi} \cdot S'_x + r_{min,x} \tag{2.8}$$

但由於推論時輸入特徵圖不如權重為定值，在訓練時同時計算式(2.9)及式(2.10)之兩參數以做為推論時的量化區間計算尺度，如式(2.11)。

$$\text{running\_max} = 0.1 \times \text{running\_max} + 0.9 \times r_{max,x} \tag{2.9}$$

$$\text{running\_min} = 0.1 \times \text{running\_min} + 0.9 \times r_{min,x} \tag{2.10}$$

$$S_x = \frac{\text{runnin\_max} - \text{running\_min}}{2^N - 1} \tag{2.11}$$

訓練完畢之量化版本模型將存有所有卷積層量化後之浮點數(w_{DQi})及所有輸入特徵圖之 running_max、running_min，於推論時一層卷積層之運算可使用以下方式。

$$\begin{aligned}\sum w_{DQi}x_i &= \sum \left(w_{Qi}S_w + r_{min,w}\right)\left(x_{Qi}S_x + \text{running\_min}\right) \\ &= \left(\sum w_{Qi}x_{Qi}\right)(S_wS_x) + \left(\sum w_{Qi}\right)(S_w \cdot \text{running\_min}) \\ &\quad + \left(\sum X_{Qi}\right)\left(S_x r_{min,w}\right) + \sum r_{min,w} \cdot \text{running\_min}\end{aligned} \tag{2.12}$$

其中$\sum w_{Qi}x_{Qi}$、$\sum w_{Qi}$及$\sum X_{Qi}$為進行整數之運算，計算完成後再分別與(S_wS_x)、$(S_w \cdot \text{running_min})$、$\left(S_x r_{min,w}\right)$進行浮點數相乘，最終將所有運算結果再與定值$\sum r_{min,w} \cdot \text{running_min}$加總得出輸出特徵圖(output feature map)之結果。而於硬體中之實作可將四個浮點數(S_wS_x)、$\left(S_x r_{min,w}\right)$、$(S_w \cdot \text{running_min})$、$\sum r_{min,w} \cdot \text{running_min}$做以下轉換。

$$n_{int} = \text{int}(n_{fp} \times 2^M) \tag{2.13}$$

將所有浮點數放大適當2^M倍後取近似整數進行運算，式(2.12)即可近似為式(2.14)。

$$\begin{aligned}\sum w_{DQi}x_i = \Big[&\left(\sum w_{Qi}x_{Qi}\right)(S_wS_x)_{int} + \left(\sum w_{Qi}\right)(S_w \cdot \text{running\_min})_{int} + \\ &\left(\sum X_{Qi}\right)\left(S_x r_{min,w}\right)_{int} + \left(\sum r_{min,w} \cdot \text{running\_min}\right)_{int}\Big] \gg M\end{aligned} \tag{2.14}$$

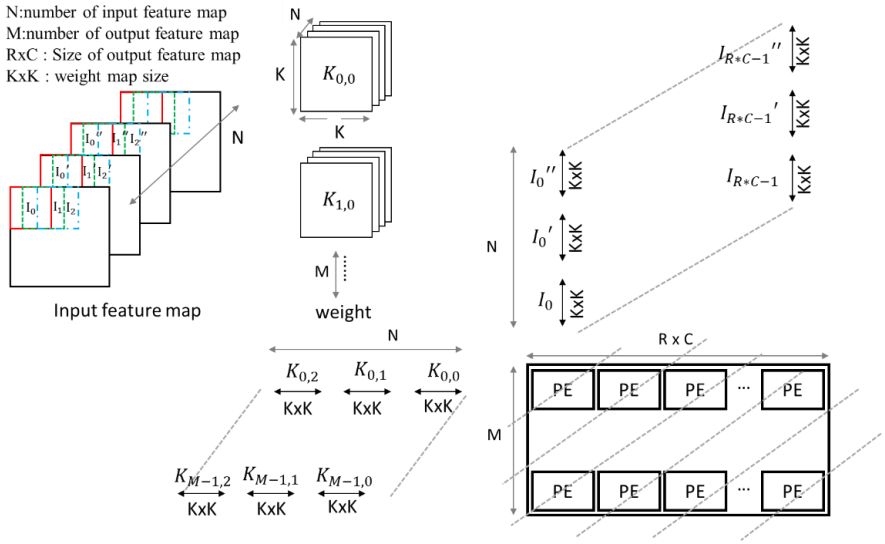
分析4bits 量化版本所有卷積層之四種浮點數後我們將M選定為8，使近似後之整數運算誤差不致過高。由上述之運算方式即可利用量化版本之參數將所有浮點數運算轉

換成整數運算，以便在硬體上加速模型推論時間。

而因為推論的計算工作有很大一部份都是在做卷積運算，卷積運算可透過重新排列資料順序轉成矩陣乘法運算，可在做矩陣乘法的同時，我們會想要能夠降低記憶體存取的時間和能源消耗，來縮減計算資源的花費。所以我們在硬體端運用脈動陣列(systolic array)的架構來做矩陣運算會是我們理想的架構。

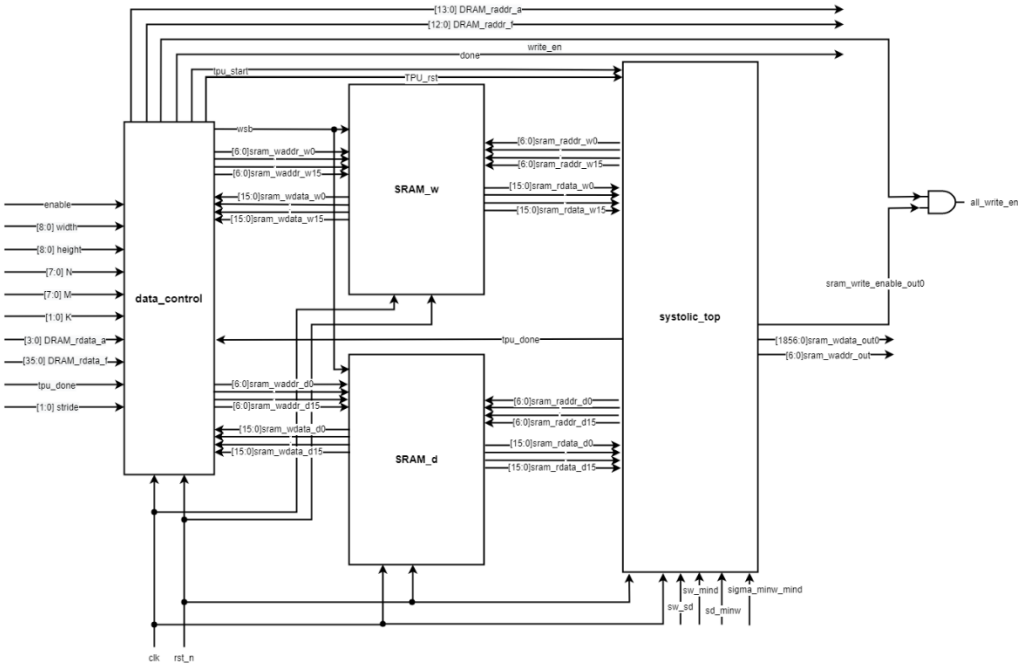
脈動陣列由多個運算單元(PE)構成，可以排列為一維、二維陣列等等結構，其特色為基本運算單元簡單，透過大量的運算單元並行運算來提高運算效率，運算單元之間只能向相鄰的運算單元傳遞資料，數據會以向下游或向右游的方向流動。這個系統能夠使一筆資料能被重複使用計算，能得到比較好的資料重複使用率。

而我們目標是實現卷積層的運算，綜合前述脈動陣列的資料流排序以及將卷積運算可透過重新排列資料順序轉成矩陣乘法運算，圖一所示，N 張輸入特徵圖與 M 組 N 張大小為 KxK 的權重圖，會輸出 M 張大小為 RxC 之輸出特徵圖。輸入特徵圖由上方送入，每一行由有同樣 receptive filed 的輸入特徵圖依序排列，而權重由左方送入，每一列為同一組權重依序排列，並將矩陣排列成平行四邊形形狀。另外，因為脈動陣列的特性，每一組輸出完成，會從左上角的對角線(灰色虛線)開始完成，往右下每個 cycle 完成一個對角線的運算單元，等所有資料通過脈動陣列，取出運算單元中的 partial sum register 的數值，便能得到輸出矩陣的數值，便可以完成卷積層的運算。



圖一 使用脈動陣列執行卷積運算的資料排列

根據上述理念，我們即可設計硬體架構，目的是輸入任意卷積層的輸入特徵圖(input feature map)以及權重(filter map)之後，利用我們所設計的架構計算出該卷積層的輸出特徵圖(output feature map)。根據前面的介紹，將卷積層的運算排列成矩陣運算後，送入脈動陣列計算，我們根據需求，利用 64x64 脈動陣列來做運算。



圖二 整體架構方塊圖

以下簡介各 block 的功能：

1. data_control：控制脈動陣列、DRAM、SRAM 之間的傳遞以及控制要放入脈動陣列的資料。
2. SRAM_w、SRAM_d：儲存目前要送入脈動陣列的輸入特徵圖(input feature map)以及權重(filter map)資料。
3. systolic_top：負責脈動陣列的計算以及寫出至輸出的 SRAM。

軟體實作結果:

表一 各版本模型測試結果(COCO test dataset)

單位：mAP

FP32(Paper)	FP32(Our)	8bits	6bits	4bits
0.758	0.759	0.756	0.753	0.711

由表一可見我們訓練出的浮點數版本模型之效能與論文所述相近，其誤差應為隨機梯度下降所造成，可忽略。而 8its 量化模型與浮點數版本仍然相當接近，6bits 版本則稍微再低一些，4bits 版本則有較大差距。然而衡量 mAP 指標之跌幅及硬體實作上之時間與空間，我們仍選擇以 4bits 版本進行硬體實作。

實際推論辨識結果

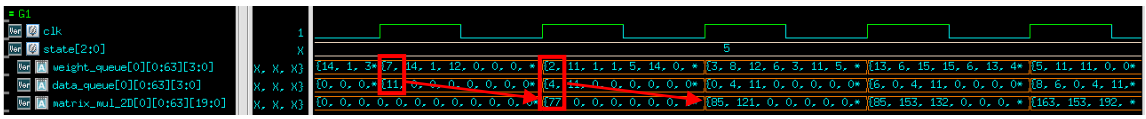
圖三 FP32版本1

圖四 4bits 版本1

由圖三及圖四可見，兩種版本之模型雖然在 COCO 測試資料集的 mAP 衡量上有大約0.05的差距，但實際演示時以肉眼分辨沒有太大差別，僅少數細節有些微不同。

硬體實作結果:

由於模型資料龐大難以測試，並且難以看出結果，此報告的實做結果以10x10x14的輸入特徵圖以及65組權重做為輸入，測試實做結果是否與目標預期相同。



圖五 脈動陣列中的運算單元$PE_{0,0}$計算波型圖

圖五中為脈動式陣列最左上角的運算單元，在已經排成平行四邊形的資料

依序送入後，從中觀察波型，weight_queue 和 data_queue 為輸入的資料，在這個 cycle 兩個資料相乘後，與 partial sum 相加，並在下一個 cycle，存入 register，第一個紅框中 weight 與 data 相乘與 matrix_mul_2D 相加，並在下一個 cycle，matrix_mul_2D 為 77，第二紅框中，weight 為 2，data 為 4，相乘過後與 matrix_mul_2D = 77 相加後為 85，在下一個 cycle 存入 matrix_mul_2D 中。

在第一筆輸出資料，也就是左上角的運算單元完成後，從 systolic_mul_outcomt 送出到 total_output_sum 中運算式 2.12，並在下一個 cycle 送出，送入輸出 SRAM 中，並以原本方式存放，之後利用 testbench 與軟體的結果比較是否相同。

在此專題中，由於模型資料龐大，無法放入 hdl 模擬中，所以我們沒有將實際整個模型的資料放入做測試，所以我們使用隨機亂數生成輸入特徵圖與權重，並以軟體計算輸出特徵圖，與硬體計算結果比較相同，以此測試這個系統架構能運用在不同的卷積層運算中。

本專題在軟體方面實作了三種不同的量化版本，其中權衡辨識效能及儲存空間、推論時間等因素後選取 4 位元版本作為硬體實作之規格。而硬體實作上以 Verilog 硬體描述語言進行模擬，主體採用的是脈動式陣列搭配其他控制電路，達成功能正確以及可使用 Verilog 模擬之效果。未來如果有機會能克服硬體空間大小的限制，期許能實際將模型量化之資料放進去脈動式陣列進行運算或是使用 FPGA 板實際演示本專題之硬體電路功能。

心得感想

起初，對於深度學習與硬體加速沒有相關的知識，在開始初期，不論是尋找適當演算法，還是處理深度學習模型都不太熟悉，起初花了一些時間學習相關的知識，並且閱讀相關的論文。開始實作專題後，也遇上了很多難題，像是如果將過大的模型放入脈動陣列等等，每一步都花上了不少時間，但非常感謝呂仁碩老師以及博士班學長的解惑，讓我們能夠處理這些難題。在這次的專題中，我們學習到了自我學習與搜尋資料的重要性，透過學習解決問題，並了解到團隊溝通合作的方式。