國立清華大學 電機工程學系

實作專題研究成果報告摘要

深度神經網路 FPGA 加速器

Deep Neural Network FPGA Accelerator

專題領域： 系統組

組　　別： B349

指導教授：鄭桂忠 教授

組員姓名：葉景元、沈沛宏

研究期間： 112年 1 月 20 日至 112 年 11月 20日止，共10個月

報告摘要

我們專題的主軸在於發展一種深度神經網路 FPGA 加速器，以解決在機器學習中由於大量重複的卷積運算而導致的效能瓶頸。考慮到 CPU 在處理這種大量平行運算時的不足，我們選擇利用 GPU 和 FPGA 進行加速運算。由於 GPU 和 FPGA 具有多核心的架構，雖然單一核心性能不如 CPU，但卻能充分發揮平行運算的優勢，從而提高運算速度。而 FPGA 可用於實現高度並行的計算架構，提供更快的訓練和推理速度。與 GPU 相比，FPGA 更節能。因此鑑於功耗的考量，我們特別選擇 FPGA 作為加速器，優化神經網路計算的硬體加速，進而提升模型的推理和訓練效能，以實現低功耗和高效能的平衡。

透過在 FPGA 上實現深度學習相關的計算，我們的目標是在維持模型準確性的同時，大幅提高運算效率並減少能源消耗。我們將深入探討硬體加速、各種不同模型的效果以及模型本身在深度學習領域的潛在優勢，包括運算速度的提升、功耗的降低以及模型整合的彈性。

我們不僅會專注於硬體層面的優化，探索如何最佳地整合 FPGA 技術，優化在神經網路辨識中使用最頻繁的捲積運算，實現對多種深度神經網路架構的有效加速；也會針對神經網路的精確度以及實現到硬體的整合深入的研究，透過軟體語言如 C++、Python，預先處理神經網路訓練與參數提取，透過多種方式提升模型準確率，如:嘗試不同優化器(optimizer)，比較多種不同神經網路架構，如:Lenet、VGG13、VGG16、VGG19、Alexnet，進行資料集預處理等方法，並在多方比較與考量下，採用 VGG16模型，而將模型訓練完成後，再撰寫硬體語言 verilog 控制 FPGA 運作，將預先在軟體部分處理好的參數儲存到 FPGA 上，結合預先處理的圖像數據，實現高效率的運算，並進行圖形的辨識。我們期望這項專題可以提供一個具有高效率、低功耗硬體解決方案，可以同時在節能和資源利用方面提升顯著的效率，同時我們也將深入挖掘 FPGA 在深度學習加速領域的潛力，為影像辨識的處理提供更快速、節能且高品質的解決方案。

報告內容

1. 研究背景與動機

現代科技日新月異，隨著人工智慧（AI）技術的飛速發展，神經網路成為解決複雜問題的重要工具。然而，深度神經網路的訓練和推理過程需要大量的計算資源，傳統的中央處理器（CPU）和圖形處理器（GPU）在處理這些任務時面臨效能瓶頸。為了克服這些挑戰，現今的研究逐漸轉向使用現場可程式閘陣列（FPGA）來實現神經網路加速器。而對於追求自動化、效率化的現代社會，圖像辨識便成為一個很重要的課題，圖像辨識技術可以應用於自動化系統，從而提高生產力和效率。自動辨識技術可以替代手動辨識和標記，節省時間和人力成本；在自動駕駛技術中也能夠使車輛感知和理解周圍環境。這對於實現安全、智能的交通系統至關重要。

神經網路是一種模仿人類大腦的計算模型，DNN 的基本單元是神經元，這些神經元組成層次結構，分為輸入層、隱藏層和輸出層。每個神經元都與前一層的所有神經元相連，並具有權重，這些權重是模型學習的參數。透過訓練過程，模型通過調整這些權重來學習從輸入到輸出的映射，實現對複雜模式和特徵的學習。

而 FPGA 神經網路加速器作為一種新興的計算硬體，展現了在解決深度學習挑戰中的巨大潛力。其可重配置性、低功耗高效能和並行計算能力使其成為應對不斷發展的神經網路應用的理想選擇。然而，仍需克服一系列挑戰，包括硬體與軟體協同開發、自適應硬體架構等方面。未來，隨著相關技術的不斷成熟，FPGA 神經網路加速器將在各個領域發揮更為重要的作用，推動人工智慧技術的更大發展。

2. 研究目的

FPGA 是一種靈活可編程的硬體，具有並行處理能力，使其適用於加速 DNN 的推斷過程。在這個背景下，深度神經網路 FPGA 加速器應運而生。這種加速器利用 FPGA 的並行計算能力，將神經網路的計算任務映射到硬體上，大大提高了模型的推斷速度。

透過硬體加速，深度學習模型在圖像辨識方面取得了顯著的性能提升，同時降低了能耗和延遲。這對於需要即時處理的應用，例如智能攝像頭、自動駕駛系統等，具有重要的意義。因此，深度神經網路 FPGA 加速器代表了一種有效的技術路徑，有望推動深度學習應用在實時場景中的更廣泛應用。

故我們目的在於實作 cifar-10資料集並以較簡單的結構、較少的參數量、節省存儲空間、同時減少計算時所需的內存和計算資源，達到較好的神經網路模型準確率和較快的圖形辨識速度。

3. 研究方法與結果

2-1. System Design

本專題實作時分為軟體以及硬體的部分，我們的流程圖可見下圖１－１。首先我們會將我們所選用的 cifar-10資料集做資料的預先處理，接著運用 vgg-16 的模型針對去 cifar-10資料集訓練並調整網路的架構，訓練完神經網路模型 fp32 後，再針對訓練好的 fp32模型量化成 int8的神經網路模型，再以 verilog 建構神經網路模型結構後應用到 FPGA 上，進行圖片辨識，以完成整體加速效果。

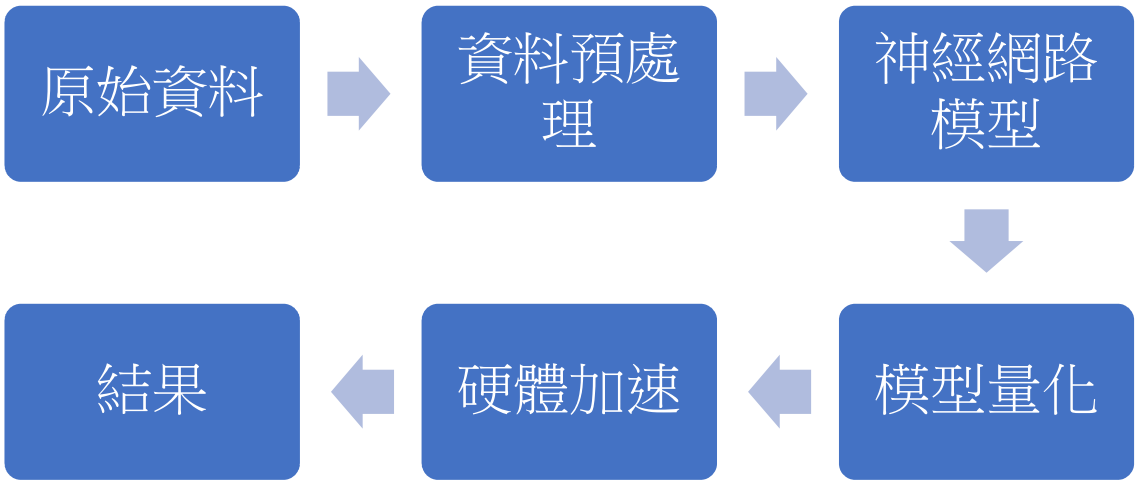

Figure 1 實作流程圖

2-2. 資料集預處理與資料增強（Data Augmentation）

在訓練 vgg16模型的過程中，因為 vgg16的模型架構相對較深，參數量較多，有過度擬合(over-fitting)的問題。因而我們需要在 vgg16模型中應用有效的正則化和防止過度擬合的機制，否則模型可能會過於適應訓練數據的噪聲(noise)和細節，而無法泛化到測試集上，使得測試集的表現不如預期。

首先，我們先利用整個訓練集算出其標準差及其平均值，並對 cifar-10的資料實作標準化，將像素值轉換為0到1之間的範圍以加快模型的收斂速度、使得模型更容易找到全局最小值、減小梯度消失或梯度爆炸的風險，提高模型的穩定性。而在於實作到我們的神經網路上的時候準確率有穩定的提升。

接著由於 vgg-16出現了過度擬合(over-fitting)的問題，在 cifar-10的資料除了標準化，又進行了隨機水平的鏡像翻轉、隨機的遮擋以及隨機的裁減後利用平均值對裁剪後的圖像進行填充。這些隨機性的變化，很好的為我們的資料更好的貼近、適應不同的視角、光照條件、旋轉等因素，在面對現實世界中的各種變化時表現較好，以提升模型的泛化能力，這樣給予更多多樣性的方法，也很好的減少了過度擬合的問題。

2-3. 訓練

2.3.1 深度神經網路架構（architecture）

我們專題採用 pytorch 工具完成網路的架構一開始採用的是 lenet 架構，其結構包含3層卷積層、2層池化層、一層全連接層，但在測試該網路和調試時，由於 lenet 較為簡單且淺的神經網路結構，在訓練資料集的表現上相當的差，而在測試集上的表現僅有約75%，故最後我們放棄使用 lenet 在 cifar-10資料集的辨識，轉而使用網路結構較深的 vgg 網路。

Vgg 神經網路分為 vgg-11、vgg-13、vgg-16、vgg-19，分別對隱藏層不斷的增加，依序分為8個卷積層、10個卷積層、13個卷積層、16個卷積層，通常以 vgg-16、vgg-19的表現較佳，我們實作上也可以發現 vgg-16、vgg-19確實相較於 vgg-11、vgg-13來的更好，其中又以16層的表現最佳，且層數也較 vgg-19較少，也在考慮到參數量對於 fpga 的負擔上捨棄了 vgg-19的使用，以 vgg-16的架構為主。

2.3.2網路架構的比較與選擇

根據上述提及的神經網路架構，比較在相同的優化器、相同的訓練次數下的對不同的網路架構、有無資料增強這兩個變因下的準確率。

Table 1各模型比較圖

	Vgg11	Vgg13	Vgg16	Vgg19	Lenet
無資料增強	78.03%	78.64%	80.40%	80.07%	64.70%
有資料增強	87.04%	87.94%	89.79%	89.64%	69.26%

可以看出在資料增強的部分很穩定提升了約5-10%準確率，對於不同的網路結構下，這樣的資料增強都幫助神經網路更加的適應測試集的不同變化和減低神經網路對於訓練集的依賴性。

而在於我們不同神經網路的選擇上，依表中我們測試的結果可以見得 Lenet 有著最簡單的架構和最少的參數量，但是 Lenet 的網路複雜度似乎難以適應 cifar-10這樣略顯複雜的資料集，故而我們放棄在 Lenet 選擇。而 vgg 神經網路在於 cifar-10資料集這種圖像辨識的工作上有著相當不錯的表現，尤其以 vgg16、vgg19這兩個網路架構在 cifar-10資料集的表現性較好，且差異甚小，但由於網路架構的考量上，希望以較簡單的網路架構和較少的參數量為主，最終還是決定以vgg16為主，加長訓練的次數並做學習率、優化器參數等細部的微調，達到最佳的準確率，最終的測試集準確率為91.29%。

2.4.3優化器(optimizer)

在訓練模型時，我們需要找到模型的最佳解，而優化器(optimizer)在這之中扮演了用來最小化損失函數的工具。而在我們的實作上，我們採用了 SGD 和 adam 兩種較常見的方式進行測試。

adam 是結合了 Momentum 和 RMSprop，並進行了偏差修正，自適應地調整每個參數的學習率，adam 在實作上通常會有優秀的表現，因而在沒有調整出好的 SGD 的參數下，我們先採用了 adam 優化器去訓練、測試我們的模型並選定架構，在我們的模型中，adam 優化器都能為我們的模型提供穩定且準確率相當高的表現。

SGD 在學習率(learning rate)調整不好的話就會造成嚴重震盪造成參數調整出錯，而學習率(learning rate)太小也會造成收斂很慢，因而這個學習率便需要自己手動的調整和測試，測試後設定在 momentum=0.9，並添加了 weight_decay

項增加正則化來防止過度擬合的狀況，此外在約0.5epochs 和0.75epochs 時，將學習率再降低0.1倍。初始階段使用較大的學習率，有助於加速模型的收斂速度。當訓練接近收斂時，降低學習率可以使模型更加穩定，避免在局部極小值附近振盪，在模型已經達到一個相對穩定的狀態時，降低學習率以微調模型。

最後我們還是以隨機梯度下降 SGD 為主要的優化器，主要原因是儘管 Adam 在許多情況下能夠加速模型的收斂、訓練的初始階段通常能夠提供更快的收斂，然而由於其自適應學習率的特性，在訓練的後期可能面臨不能收斂到全域極值點的問題。這是因為 Adam 在訓練過程中根據每個參數的動量和二次動量自動調整學習率，但有時這種自適應性可能使得學習率變得過小，導致模型在搜索空間中過早停滯。所以轉而使用具有手動調節學習率功能的隨機梯度下降 SGD 優化算法，SGD 允許更靈活地調整學習率。這樣的操作提供了更大的控制權，有助於避免過度縮小學習率可能帶來的問題，同時在訓練的後期更有可能找到全域極值點。

2-4. 深度神經網路模型量化(Quantization)

模型量化是一種優化深度學習模型的技術，目的在減少模型的存儲需求和計算成本，同時保持模型在某種程度上的性能。這一過程主要包括兩個方面的優化：模型權重的量化和激活值的量化。權重量化中，通常將原來的浮點數權重轉換為較低位數的定點數或整數表示。將32位的浮點數權重轉換為8位的整數表示，因為低位元數的表示意味著更快的運算速度，這減少了記憶體頻寬的需求，並且更容易在硬體上進行平行計算。但是降低位元數通常伴隨著精度的損失。然而，對於一些應用而言，這種精度損失是可以接受的。對於影像分類任務而言，8位的精度足夠應對大多數情況。這樣8-bit_signed_integer 的壓縮可以大大減小模型的大小，節省存儲空間，同時減少計算時所需的內存和計算資源。同時我們也會對模型的激活值進行量化，這樣可以減少模型在推斷過程中的計算量，同時提高模型的運行速度。而這樣的模型量化在後續的硬體去建構我們的網路時，能夠很好的減低 fpga 的負擔和運算量，故而在訓練完模型後進行了模型量化的部分。模型量化可以分為訓練中量化（Training-aware Quantization）和訓練後量化（Post-training Quantization）兩種量化策略。

訓練中量化（Training-aware Quantization）在模型訓練的同時，根據模型的動態範圍和權重分佈等信息，有助於量化算法更好地適應模型的特性，減小精度損失。由於模型在訓練中考慮了量化，有助於模型更好地收斂到在量化情境下的最優解。但因為需要在每個訓練步驟中考慮量化，訓練時間相對較長、也需要更多的計算資源。

訓練後量化（Post-training Quantization）由於量化是在模型訓練完成後進行的，整個訓練過程不需要考慮量化，因此訓練時間相對較短，但整個訓練過

程更簡單。但相對的量化是在模型已經訓練完成的情況下進行的，可能無法充分適應模型的特性，因此精度損失可能較大。

在此專題選擇以訓練後量化（Post-training Quantization）作為較簡易的量化方式，但也相對地造成精度的下降比較明顯。

2-6. 硬體加速

2.6.1 資料取用與整體架構

當我們使用 python 將 VGG-16神經網路訓練完成之後，將訓練結束後得到的參數與 scale 值提出並從 fp32量化(quantize)成 int8，以減少模型運算複雜程度與規模，而因為 VGG-16的參數量較龐大，我們無法一次將這麼龐大數量的參數投入運算，因此我們先將參數與輸入資料儲存在 FPGA 中的 SRAM 裡面分批取用，當一層 convolution 層運算完，將結果傳遞至下一層之後，再取用下一層的參數，以時間換取空間，避免矩陣規模過大超出 FPGA 承受範圍，因此硬體結構如下所示:

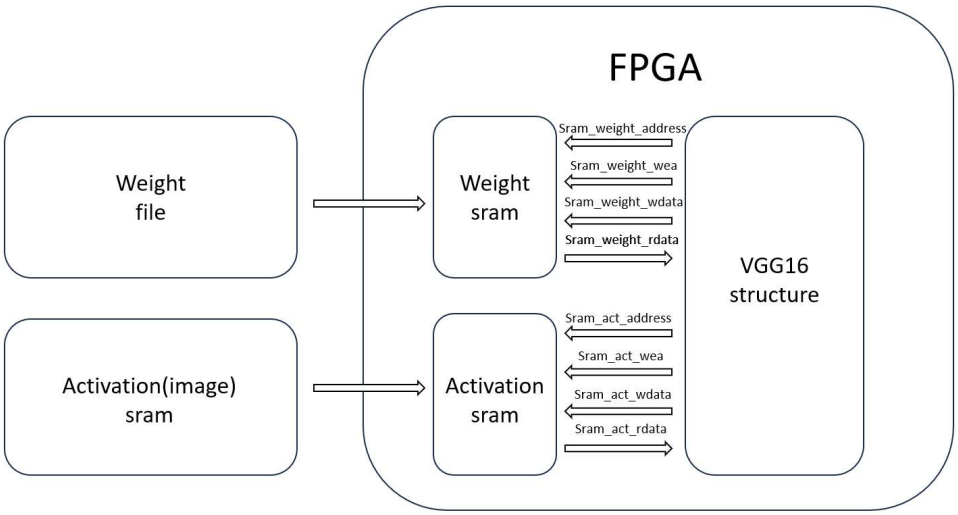

Figure 2 整體架構圖

2.6.2 VGG-16 structure

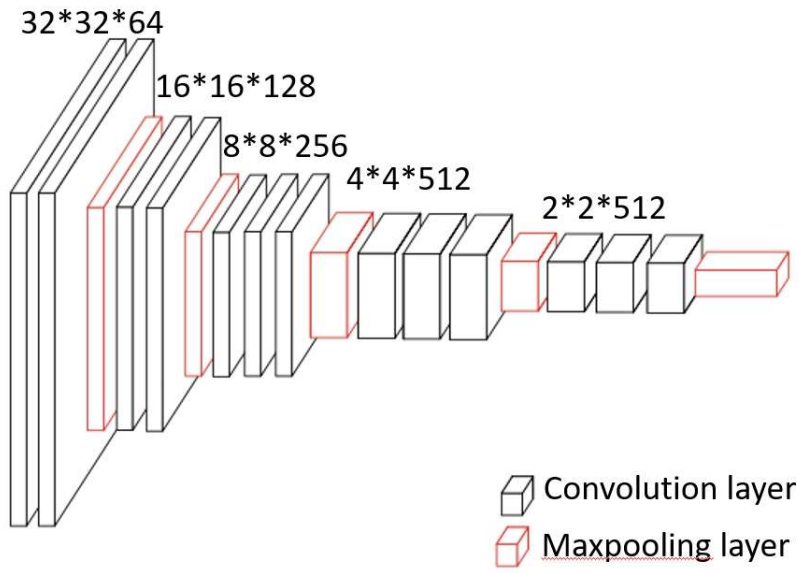

Figure 3VGG-16模型

而由2.4.1我們已知 VGG16的結構有13層 convolution、與5層 maxpooling 層，其中 convolution 層會持續使用到 convolution 運算，因此在設計硬體時我們將 convolution 層中的捲積功能模組化設計，並優化其功能以達到效率的提升。

2.6.3 convolution structure

在單一層 convolution 中，我們選用兩個 single filter layer 進行並行運算，即同時運算兩顆卷積核以增加效率，當 weight 從 SRAM 進來之後先經由卷積核(filter)將該次運算所需要的 weight 存下來後再分到各個 single filter layer 中的 convolution unit(CU)進行卷積運算，而在各 single filter layer 中，RF selector 將卷積核覆蓋的影像區域(稱為視窗)資料對應傳送給14個 CU，一個 CU 大小即視窗大小(3*3(VGG16的卷積核大小)*8(bit))，而在 CU 中，我們執行乘法與累加，再進行 Quantize 將計算結果從16bit quantize 回 int8，以利傳至下一層進行運算，並避免梯度消失或爆炸，也避免 int8在卷積過程中 bit 數越來越大最終超過硬體限制，我們將在接下來介紹 CU 結構。

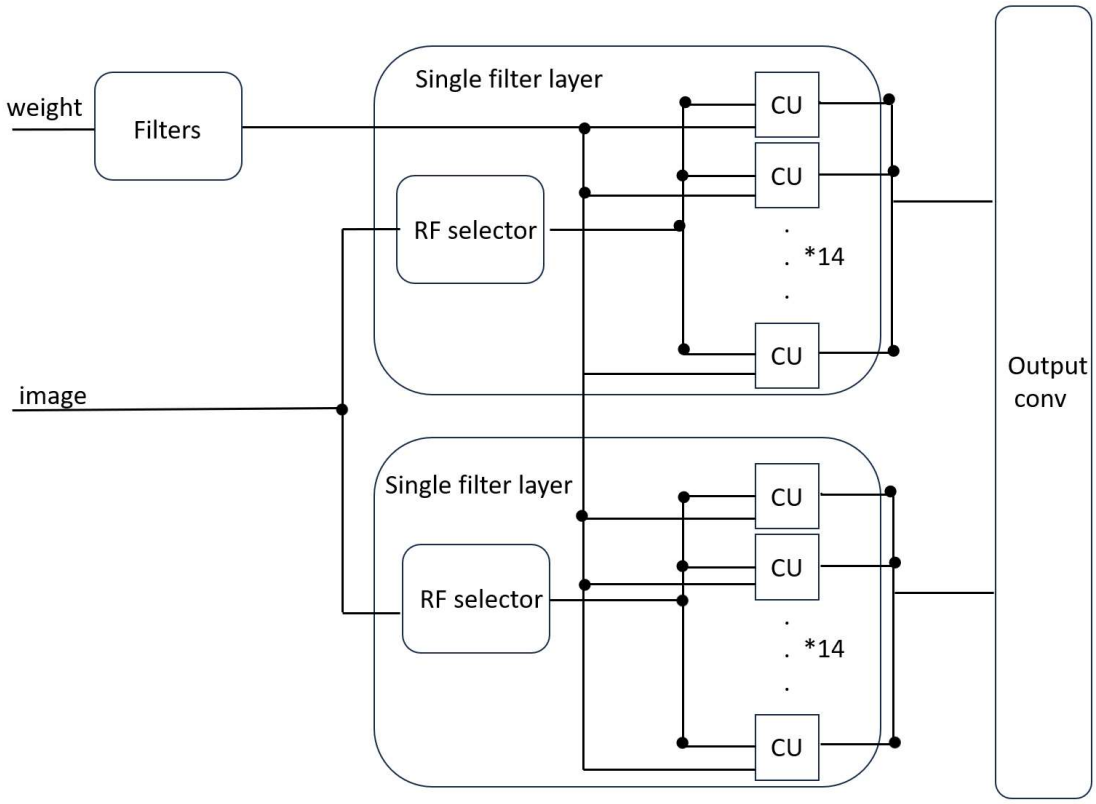

Figure 4Convolution Layer 結構

2.5.3 Convolution Unit(CU)

在 convolution unit 中，我們除了進行卷積之外，還會進行 weight 的縮放與數值的 quantization，以避免數值的消失或爆炸。由於 int 的乘法在每次進行運算之後會從 n bit 變成2n bit，而加法會從 n bit 變成 n+1 bit，而在 convolution unit 中我們正需要進行多次的乘法與累加運算，如若沒有進行處理，在進行幾次 convolution 後，數字的 bit 數將會達到難以預計的數目，更別提在 convolution 運算完之後還要乘上 scale 讓數字進一步的放大，因此我們需要在 CU 中對結果進行縮放與處理。

在單一 CU 中，我們將 input 的兩個 int8的數字 signed extension 成 int16，再使用移位加法器執行 int16的乘法，結果為32bit，並且將此32bit 的數字限制在-32768~32767，若超過上下限則限制在最大與最小值，再進行累加以完成 convolution。

完成 convolution 後的結果，將再傳至 quantize 中，將該層的結果全部乘以 scale 後再右移16bit 將最後的結果 clamp 在127~-128之間，這樣最後的 output 就會又恢復成 8bit 並成為下一層的輸入。

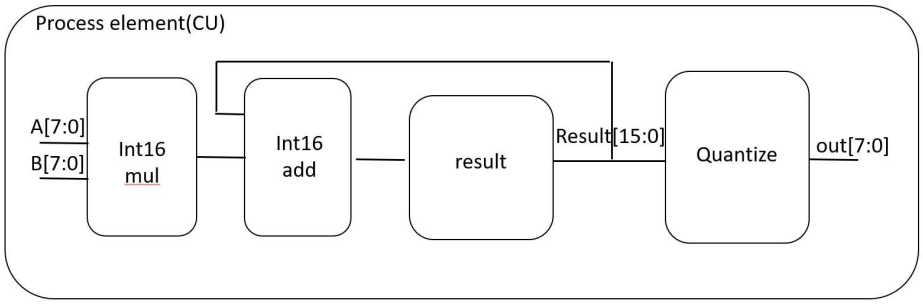

Figure 5 Convolution Unit 結構

4. 心得感想

這次專題於我們而言是個很難得的經驗，從去年選擇做這個專題的時候對於神經網路與機器學習的一竅不通，甚至可能連聽說都沒聽說過，到現在我們可以自己訓練網路並建立起神經網路模型，在這一年的過程我們學習了許多與機器學習有關的知識，查看了相關的論文，也去學習了機器學習導論、深度機器學習開放式課程等與機器學習有關的課程，如果沒有這個專題我想我們也不會對神經網路有如此深入認識與了解。

在做這個專題的過程中，我們也遇到了許多困難，像是:一開始在要訓練神經網路時，我們對於訓練神經網路這件事一點概念都沒有，連環境建構都摸索許久，在建構過程中也遇到不少問題，但我們靠著自己上網查找資料將困難一一排除。在進行 quantization 時遇到不少 bug、訓練神經網路時遇到準確度瓶頸或是在建構網路架構時也遇到許多問題，多虧了陳彥文助教，在我們遇到問題時不厭其煩地幫我們解答，也提供許多有用的參考資料與文獻使我們在進行專題研究之前，了解了目前深度學習加速器的相關研究。這些研究主要涉及模型的量化、裁剪、網路設計等方面。深入瞭解前人的經驗對於我們專題的設計和實現提供了寶貴的參考。雖然到最後還是有部分困難沒有解決，但我們在這次專題中學到更多的是自行蒐集資料以及解決問題的能力，我想這是我們在這次專題過程中最寶貴的收穫

通過這次專題研究，我深刻體會到深度學習在硬體層面的挑戰和機遇。FPGA 加速器的應用為深度學習的發展開啟了新的可能性，同時也讓我對硬體設計和深度學習有了更深的理解。我相信這次經歷對我們未來的學術和職業生涯都將有著深遠的影響。