基於 PYNQ 和高階合成的物體辨識功能快速開發流程
Rapid Development Process for Object Recognition Based on PYNQ and High-Level Synthesis

組員：陳帛駿

摘要

在當今日益智能化的世界中，物體辨識技術已經成為了一個具有革命性潛力的領域。然而，實現高效的物體辨識並不總是一個簡單的任務，它需要先進的硬體和複雜的演算法結合，以滿足不斷增長的需求。

在這一背景下，FPGA 技術嶄露頭角，為物體辨識領域帶來了嶄新的機遇。FPGA 以其卓越的計算能力和低功耗特性，成為了實現高性能物體辨識模型的理想硬體平臺。

然而傳統 FPGA 開發過程複雜，因此現今主流的 AI 依然以 GPU 作為其計算核心。本研究旨在深入探討如何以 PYNQ-ZU 開發板和高階合成（High-Level Synthesis，簡稱 HLS）軟體加速 FPGA 的開發週期。

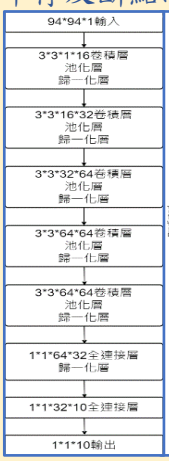
研究過程

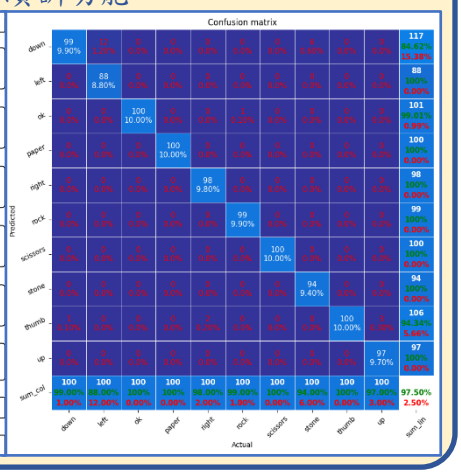
1. 在獲取資料集後，以 GPU 訓練卷積神經網絡（CNN）模型。
2. 以 C++開發各功能並使用 Vitis HLS 軟體將其轉為 Verilog。
3. 使用 Vivado 軟體將 Verilog 以 IP 核的形式集成到 FPGA 核心中，並導出 bitstream 檔。
4. 將檔案燒錄至 PYNQ-ZU 開發版，並根據分配的記憶體位置編寫相應的 python 驅動。
5. 配置參數記憶體空間並導入。
6. 最後使用驅動執行推論。

訓練方法

訓練集為十種手勢各 500 張，驗證集和測試集為各 100 張。為了避免在訓練中產生過擬合現象加入了學習率調整函式在一定 epoch 後降低學習率。除此之外還加入了早停及斷點續訓功能。

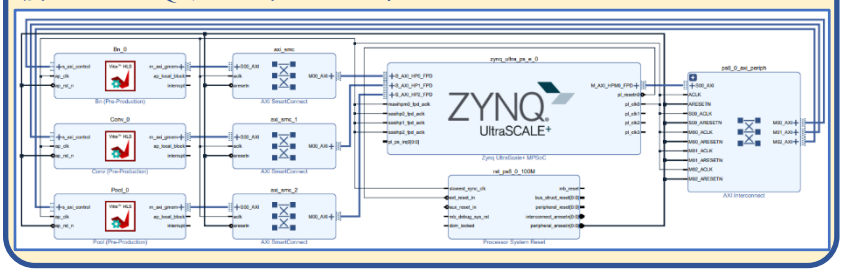
在我的模型中我將卷積核的權重二值化為 1 和-1 以減少權重的儲存並提升計算速度。由於二值化能看作是一種非線性化操作，因此我沒有使用常見的激勵函數，而是使用了批歸一化層以提高訓練的穩定性和速度。





IP 整合

這個步驟主要是將各個 IP 核與開發版的 ZYNQ 核心透過 AXI 連線、整合並導出 bitstream 檔供開發版燒錄。主要思路是將 ZYNQ 核心的 master 端透過中繼分別接入三個 IP 核的 slave 端，並將三個 IP 核的 master 端分別透過中繼接入 ZYNQ 核心的 slave 端。



驅動

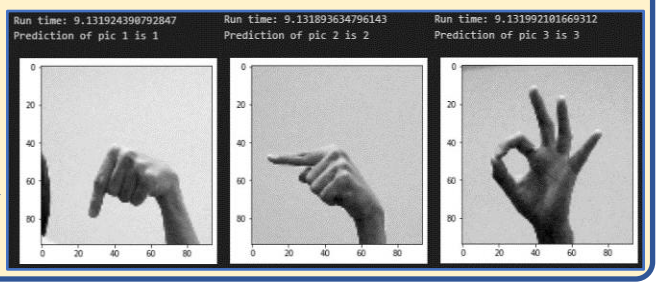
為了調用 FPGA 執行任務，必須要用 python 為 PS 端開發驅動與 PL 端進行互動。此為 PYNQ 系列開發版的特點。

在開發驅動時還必須參照 HLS 生成的 verilog 為每個 I/O 端口配置相應的記憶體位置。

```python
def RunConv(conv,Kx,Ky,feature_in,W,bias,feature_out):
    conv.write(0x10,feature_in.shape[2])
    conv.write(0x18,feature_in.shape[0])
    conv.write(0x20,feature_in.shape[1])
    conv.write(0x28,feature_out.shape[2])
    conv.write(0x30,Kx)
    conv.write(0x38,Ky)
    conv.write(0x40,feature_in.physical_address)
    conv.write(0x4c,W.physical_address)
    conv.write(0x58,bias.physical_address)
    conv.write(0x64,feature_out.physical_address)
    conv.write(0, (conv.read(0)&0x80)|0x01)
    while not ((conv.read(0)>>1)&0x1):
        pass

def RunPool(pool,feature_in,feature_out):
    pool.write(0x10,feature_in.shape[2])
    pool.write(0x18,feature_in.shape[0])
    pool.write(0x20,feature_in.shape[1])
    pool.write(0x28,feature_in.physical_address)
    pool.write(0x34,feature_out.physical_address)
    pool.write(0, (pool.read(0)&0x80)|0x01)
    while not ((pool.read(0)>>1)&0x1):
        pass

def RunBn(bn,feature_in,W,bias,feature_out):
    bn.write(0x10,feature_in.shape[2])
    bn.write(0x18,feature_in.shape[0])
    bn.write(0x20,feature_in.shape[1])
    bn.write(0x28,feature_in.physical_address)
    bn.write(0x34,W.physical_address)
    bn.write(0x40,bias.physical_address)
    bn.write(0x4c,feature_out.physical_address)
    bn.write(0, (bn.read(0)&0x80)|0x01)
```

結論

有成功調用 FPGA 進行推論，但執行時間不如預期。並且有不少圖片的推測結果與 GPU 有出入。經分別檢查單層輸出發現，有少數地方出現不足 0.00001 的誤差，經過多層的運算後，誤差可能會被放大到不可忽視的程度，猜測是因不同的計算精度或對浮點數運算的差異所導致。