Deep Neural Network FPGA Accelerator

深度神經網路 FPGA 加速器

專題領域 ： 系統領域
組　　別 ： B128
指導教授 ： 鄭桂忠
組員姓名 ： 廖威竑、郭柏辰

Abstract

在機器學習中，因為計算方式有大量重複處理的卷積運算(Convolution)，因為 CPU 不擅長處理大量平行運算，在此不是理想的運算單元，因而有使用 GPU、FPGA 加速運算的應用。由於 GPU 和 FPGA 的架構中都具備多顆核心，儘管單一核心性能不比 CPU，卻可以利用其平行運算之特性，使運算速度提升。考量到功耗問題，FPGA 可以同時實現低功耗且加快驗證速度，此專題中我們選用 FPGA 做為加速器。

在此專題中，我們結合深度學習的方法，針對圖片進行超高解析(Image Super Resolution)，藉由輸入低解析度(Low Resolution)的圖片，經 FPGA 中的深度學習模型運算後，得到高解析度(High Resolution)的圖片。並比較「經 FPGA 運算」與「全部經 CPU 運算」之所需時間，並驗證兩張圖片的一致性，達成加速之成效。

所使用之模型為 Residual Dense Network (RDN)，藉由 Vivado、DNNDK、PetaLinux 等工具將模型轉化成 FPGA 可應用之形式，並且利用軟體語言，如 C++、Python，控制 FPGA 板之運作。

Introduction

(一). 前言

過去要將低解析度的圖片轉變成高解析度時，大多採取插值法，也就是將該點的鄰近像素點取樣並做平均作為該點的值。常見的方法為最近鄰差值、雙線性插植法、雙三次插值法等等。其缺點為在物件邊緣處往往容易失真，且對於較大型圖片的運算速度較慢。而利用深度學習的方法則可以藉由非線性的激勵函數(activation function)，較精確的找出圖片中物件之邊界，並且因為運算大多為單純的矩陣加法與乘法，可以利用特別設計的演算法，提高運算效率。因此在此專題中我們採取深度學習的方法，並透過 FPGA 加速圖片運算之速度。

(二).模型分析

我們此次使用的 FPGA 版本為 Xilinx TUL PYNQ™-Z2 board。其內部核心晶片型號為 ZYNQ XC7Z020-1CLG400C，是一塊以 Python + Zynq 為概念的開發板。板子上可以分為 PS 和 PL 兩個部分。

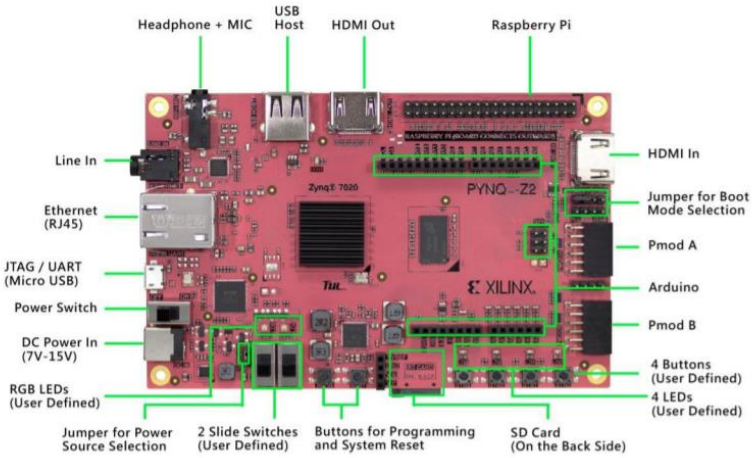

圖 1 Xilinx TUL PYNQ™-Z2 board

殘差(Residual)網路在每經過幾層就給予一個捷徑連結(Shortcut Connections)，可以確保較深網路訓練必不遜於較淺層網路。因此在專題中我們選擇使用 Residual Dense Network (RDN)模型，以確保結果的穩定性。下圖為其網路架構。

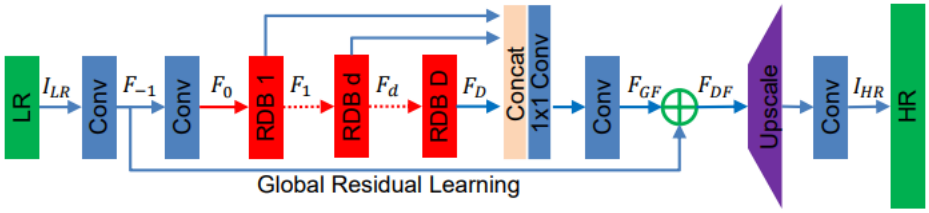

圖 2 Residual Dense Network

在 Residual Dense Blocks (RDB)中，每一層 feature map 皆會與其後方每一層 feature map 連接，傳統上 L 層的網路架構只會存在 L 條線連接，但在此架構模型之下，可以產生$L(L+1)/2$ 條線，在文中也提到此作法可以有效減少所需要訓練的變數、減緩梯度消失(vanishing-gradient)以及增加變數重複使用率。

(三).分析方法

我們針對兩面向分析經運算出的圖片 :(1) 運算速度 (2) 成像品質

速度方面利用 Colaboratory 運行模型，並得到運算後之圖片作為「單純使用 CPU 運算」之結果；而將「使用 FPGA 運算」之結果與之對比，並計算加速成效。成像品質方面我們

利用 PSNR(Peak Signal and Noise Ratio)與 SSIM(Structural Similarity Index) [4]兩種分析方法分析。PSNR 分析待測圖片與參考圖片之間受雜訊干擾的比率，單位為 dB。方法是將兩張圖片的每個像素點做相減的總和，並與圖片的最大值比較。

$$PSNR = 10\log\left(\frac{MAX_I^2}{MSE}\right) = 10\log\left(\frac{255^2}{\frac{1}{N}\sum_{i=1}^{N}\left(I(i) - \hat{I}(i)\right)^2}\right)(dB)$$

$I(i)$： pixels of high resolution , $\hat{I}(i)$：pixels of super resolution

SSIM 則是比較待測圖片與參考圖片之間的平均、標準差與共變異數，藉由較整體性的比較分析兩張圖片在結構上產生的差異。

$$SSIM(I,\hat{I}) = [\mathrm{C}_l(I,\hat{I})]^{\alpha}[\mathrm{C}_c(I,\hat{I})]^{\beta}[\mathrm{C}_s(I,\hat{I})]^{\gamma}$$

$C_l = \frac{2\mu_I\mu_{\hat{I}}+C}{\mu_I^2+\mu_{\hat{I}}^2+C}$ (mean), $C_c = \frac{2\sigma_I\sigma_{\hat{I}}+C}{\sigma_I^2+\sigma_{\hat{I}}^2+C}$ (standard deviation), $C_s = \frac{\sigma_{I\hat{I}}+C}{\sigma_I\sigma_{\hat{I}}+C}$ (covariance)

α, β, γ : parameters for controlling importance.

Methods

1. 訓練 DPU 支援之模型，考量到 DNNDK、DPU 支援的版本，我們在此以 Tensorflow v1.12.0、Keras v2.2.4 在 Ubuntu 16.04 環境下進行訓練。
2. 儲存模型為 hdf5 檔案，利用 keras_2_tf.py 檔案轉存為 checkpoint 和 pb 檔案以提供 freeze model 使用。
3. Freeze graph，將模型內訓練所用之變數轉化為常數儲存為 pb 檔案。
4. Quantization，將訓練所得之權重 (weights) 由 32 bits 浮點數轉換為 8 bits 整數儲存。以提供後續 Vitis 工具使用。
5. Compile，根據給定的硬體描述檔案 dcf 檔以及 quantized 模型產生 elf 檔，用以在 FPGA 板內執行。

將 PS 端要處理的程式(C++)與 DPU 所需要的 elf 檔做 cross-compile，最後生成可執行的 binary 檔案，可以透過 shell 來執行。

```
DNNC Kernel topology "RDN_44_kernel_graph.jpg" for network "RDN_44"
DNNC kernel list info for network "RDN_44"
                               Kernel ID : Name
                                       0 : RDN_44

                             Kernel Name : RDN_44
--------------------------------------------------------------------------------
                             Kernel Type : DPUKernel
                               Code Size : 1.07MB
                              Param Size : 15.59MB
                            Workload MACs : 63265.70MOPS
                          IO Memory Space : 4.51MB
                               Mean Value : 0, 0, 0,
                               Node Count : 269
                              Tensor Count : 270
                 Input Node(s)(H*W*C)
                  F_m1_convolution(0) : 44*44*3
                Output Node(s)(H*W*C)
                    SR_convolution(0) : 88*88*3
```

圖 3 RDN 44 * 44 執行結果

Results

圖 4 左側部分為此次實驗中用來測試的高解析度圖片，右側則為經過 FPGA 估計所得圖片。由於 DPU 記憶體空間無法將整張圖片讀入，在運算時我們採用將圖片分割成小區塊(Patches)，再將 Patches 再次接合的方式實作。以單一區塊而言，處理後的圖片邊緣較為銳利，亮度也會有所調整；且在接合後整張圖片會變成區塊分布，造成圖片部分失真。

圖 4 (左)原高解析圖片、(右)運算後 SR 圖片

表 1 呈現經 DPU 運算得到的結果。由結果得知每個 Patch 約需 0.46 秒，而對於類似大小的圖片的總運算結果約為 180 秒，也就是 3 分鐘，並得到一張長寬皆放大兩倍之 SR 圖片。

表 2 比較使用(左) FPGA 的 DPU 運算時間，以及(中)使用 Colaboratory 所提供的資源，並將其使用的運算單元設定在使用 CPU，在相同模型下運算相同大小圖片所需時間。並計算 Speedup (右)。透過 FPGA 後約可節省 60%以上的運算時間。

表 1 RDN 44 x 44 實驗數據

Photo	Width	Height	#Patches	Average Patch Time (s)	Time(s)
0001	1020	702	384	0.4602	176.71
0002	1020	924	528	0.4599	242.84
0003	1020	678	384	0.4602	176.72
0004	1020	672	384	0.4599	176.60
0005	1020	804	456	0.4602	209.83
0006	1020	678	384	0.4600	176.62
0007	1020	678	384	0.4600	176.65
0008	1020	678	384	0.4601	176.68
0009	1020	762	432	0.4601	198.77
0010	1020	822	456	0.4602	209.84

表 2 使用 FPGA 與 CPU 之比較

Photo	FPGA Time(s)	Intel(R) Xeon(R) CPU @ 2.20GHz Time(s)	Speedup
0001	176.71	531.99	66.78%
0002	242.84	672.90	63.91%
0003	176.72	502.95	64.86%
0004	176.60	492.07	64.11%
0005	209.83	596.01	64.79%
0006	176.62	505.63	65.07%
0007	176.65	506.02	65.09%
0008	176.68	505.47	65.05%
0009	198.77	567.01	64.94%
0010	209.84	609.43	65.57%

表 3 PSNR、SSIM

Photo	PSNR(dB)	SSIM
0001	28.066	0.672
0002	28.346	0.584
0003	27.833	0.694
0004	27.976	0.765
0005	28.588	0.801
0006	28.178	0.695
0007	28.109	0.590
0008	27.978	0.671
0009	27.867	0.670
0010	28.439	0.708

表 3 呈現運算出的圖片與原高解析圖片之差異。PSNR 運算的結果皆在 27~28 之間，代表峰值訊號與雜訊約差 1000 倍，人眼判斷上並沒有看到雜訊的干擾。在結構分析(SSIM)上則數值相差較大，其原因推測為整體亮度經運算後變暗，與原圖相差較多；另外在 Patches 連接出仍有細微的黑色條紋，也是造成 SSIM 數值較低的原因之一。

Conclusion

1. 利用 FPGA 上的 DPU 進行圖片的運算，可以將運算速度提升約60%。對於單一 Patch 而言，CPU 因為擁有較高的操作頻率，速度高於 DPU 運算，因此較小張的圖片加速成效較不明顯；對於較大型的圖片而言，FPGA 加速效果顯著，CPU 則因為無法進行有效的平行運算，運算速度大幅上升。
2. 圖片在經過運算後沒有產生過多雜訊，但在結構性的問題上產生誤差。經過 FPGA 運算之圖片會產生整體亮度較暗的問題；在 Patches 的連接處有相接時的黑色線條。最終成像品質的問題。

Feedback

感謝鄭桂忠教授給予我們機會，學習深度學習相關領域，並最終完成專題。感謝鄭揚翰與李兆紡學長在學與暑假期間給予專題建議與解決方向，一路引領我們向前。感謝組員，在專題期間能夠共同解決各種技術問題與討論可行方案。雖然專題中有些想法而未能實現，但藉由這次機會也學到不少智識與技巧，希望在未來漫長的研究生涯中能夠步步向前不再留下遺憾。