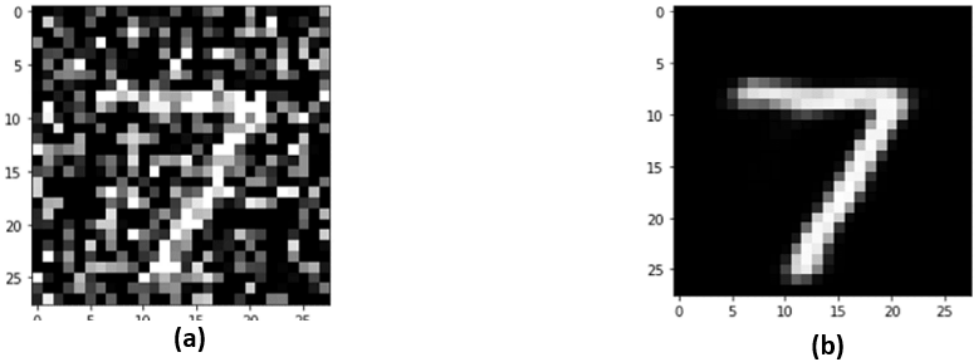
VLSI System Design Final Project Group Report – Group 12

Project Title: Convolutional Neural Network Accelerator for Image Denoising	
Name: 陳奕帆	ID: 110062612
Name: 小山翼	ID: 110062526

Project Description:

1. 應用主題

In this project, we apply DNN accelerator on one of the fundamental challenges in the field of image processing and computer vision, image denoising on MNIST. The picture below roughly explained the goal in image denoising, it is to estimate the original image (a) by suppressing noise from a noise-contanimated version of the image (b). This technique plays an important role in a wide range of applications such as image restoration, visual tracking, image registration, image segmentation, where obtaining the original image content is crucial for strong performance **[1]**.



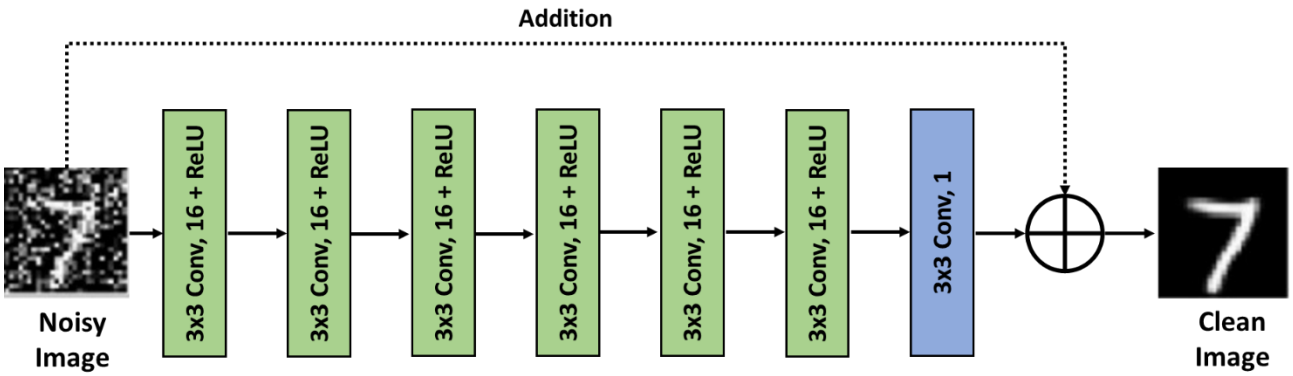
Example of image denoising

2. 軟硬體架構規劃

Software

A. Denoise CNN Model Architecture

We design a CNN model based on this paper**[3]** for image denoising. The model architecture is showed in below figure. The model composed of 7 layers. The first six layers are 3x3 convolution layer with 16 channels and ReLU activation function. The last layer is 3x3 convolution layer with 1 channel. Besides, we use residual learning to add the input noisy image with the last layer's output feature map. Besides, different from the existing discriminative denoising models which usually train a specific model for additive white Gaussian noise (AWGN) at a certain noise level, our DnCNN model is able to handle Gaussian denoising with unknown noise level (i.e., blind Gaussian denoising).



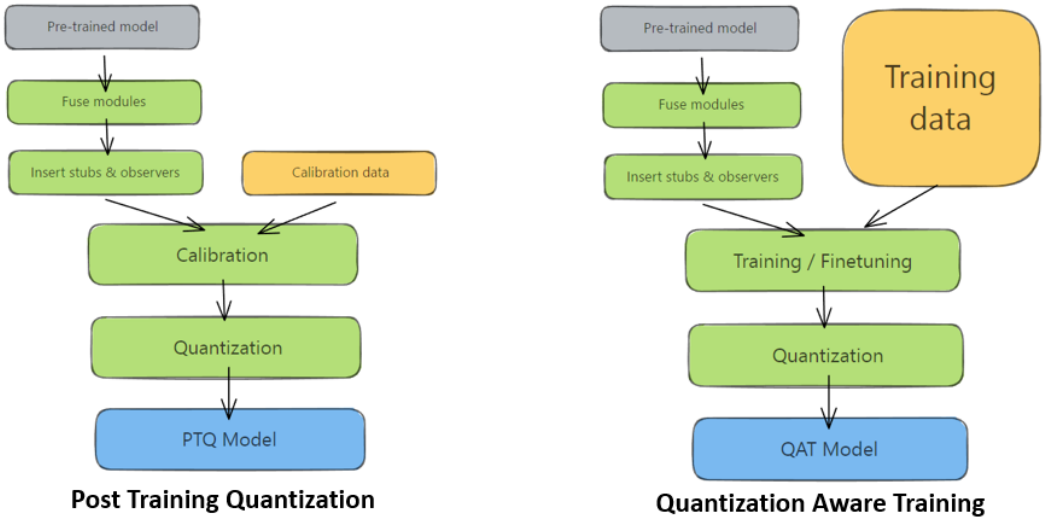
Denoise CNN architecture

B. Residual learning

Consider a clean image $\mathbf{x}$ corrupted by noise $\mathbf{v}$. Noisy image $\mathbf{y} = \mathbf{x} + \mathbf{v}$. Our goal is to recovering $\mathbf{x}$ can be formulated as finding a parametric function $\hat{\mathbf{x}} = \mathbf{F}(\mathbf{y}; \boldsymbol{\theta})$. Instead of directly learning $\hat{\boldsymbol{x}}$, we are going to learn the residual mapping $\mathbf{H}(\mathbf{y}; \boldsymbol{\theta}) \approx -\mathbf{v}$. Therefore, the denoising parametric model based on learning can be reformulated as $\boldsymbol{F}(\boldsymbol{y}; \boldsymbol{\theta}) = \boldsymbol{H}(\boldsymbol{y}; \boldsymbol{\theta}) + \boldsymbol{y}$. Several studies show that the residual mapping is much easier to be learned than the original unreferenced mapping. Therfore, we use residual path in our model to improve training speed and the performance. With the residual learning strategy, DnCNN implicitly removes the noise in the hidden layers. This property motivates us to train a single DnCNN model to tackle with several general image denoising tasks such as Gaussian denoising, single image super-resolution and JPEG image deblocking.

C. Quantization

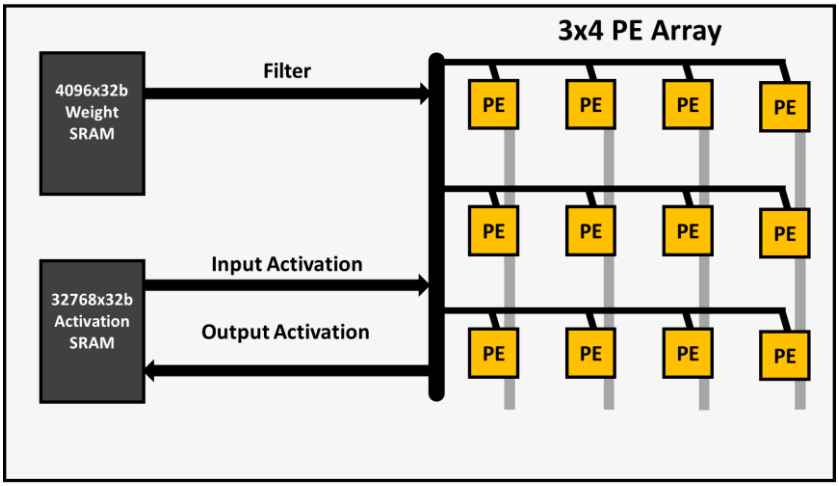
We will do Post Training Quantization (PTQ) since we have to map the computaion result between -1 and 1 to -128 and 127. To reduce the Mean Square Error (MSE) after PTQ, we apply Quantization Aware Training (QAT) on the DnCNN model. QAT will train DNNs for lower precision INT8 deployment, without compromising on accuracy, and this is achieved by modeling quantization errors during training which helps in maintaining accuracy as compared to FP16 or FP32 **[2]**.

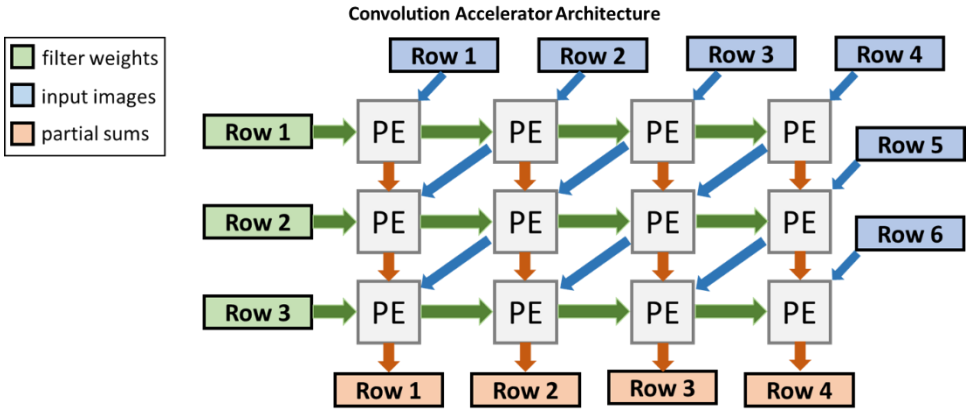


Hardware

A. DnCNN Accelerator Architecture

Our accelerator is consisted of one 4096x32b dual port weight SRAM, one 32768x32b dual port activation SRAM, and one 3x4 PE Array. We use the Eyeriss row stationary method**[5]** in the DnCNN accelerator to maximize data reuse. The accelerator consists of a 3x4 PE Array, which can process convolution of 6x6 input feature map with 3x3 kernel at the same time. The row stationary method divide 2D convolution into multiple 1D-primitive convolution. Each PE calculates 1D-primitive convolution and forward the partial sum to the PE below. Accumulate the result of each PE column resulting in a 3x3 2D convolution. Since the PE Array has four column, we can generate four rows of output feature map at the same time. Most of the previous work focus on certain type of data reuse. Either reuse input activations, filters, or output activations. However, the row stationary method reuse input activation, filters, output activations simultaneously. Filters are pass to the first PE column, then forward to the next PE column. Therefore, filters are reuse horizontally. Input activations are pass to each diagonal because filters will slide from top to down in the convolution. Therefore, input activations are reuse diagonally. Since each PE only caculate 1D convolution, we need to accumulate multiple 1D convolution to generate 2D convolution. Hence, each PE will pass the partial sum result to the PE below. Then, we accumulate each PE's partial sum results. Therefore, output activations are reuse vertically. With the row stationary method, we can maximize the data reuse to avoid many memory access. Generate a high energy-efficient DnCNN accelerator.

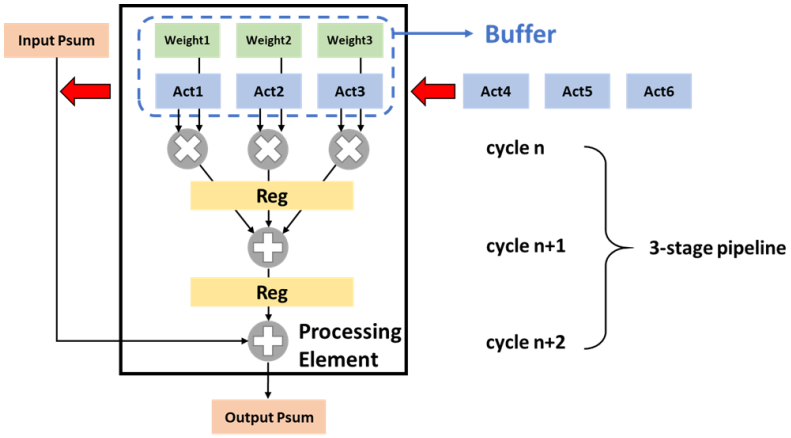




3x4 PE Array

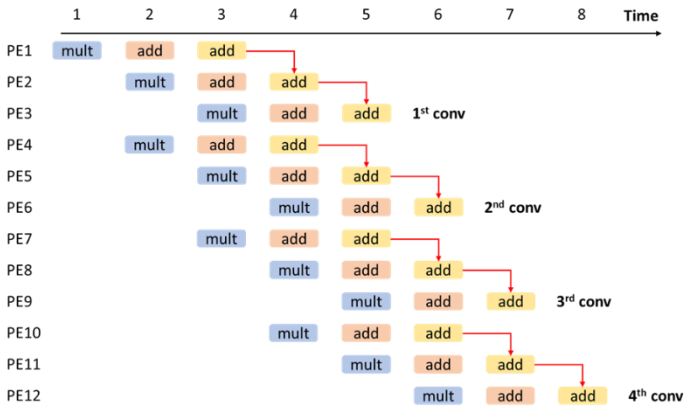
B. Pipelined DnCNN Accelerator

We pipelined the proceesing element into three stage to reduce clock cycle time. Each PE contains a buffer storing three weights and three input activations. The weights stay in the buffer for data reuse. The input activations is an FIFO buffer. At each cycle, it will pop one activations from the front and push one activation to the last. Because the kernel will slide from left to right and the weight remain stationary in the buffer, we need to shift activations to left at each cycle. Although pipelined DnCNN Accelerator can increase clock frequency, it needs registers to store internal results. Therefore, it will increase the total cell area.



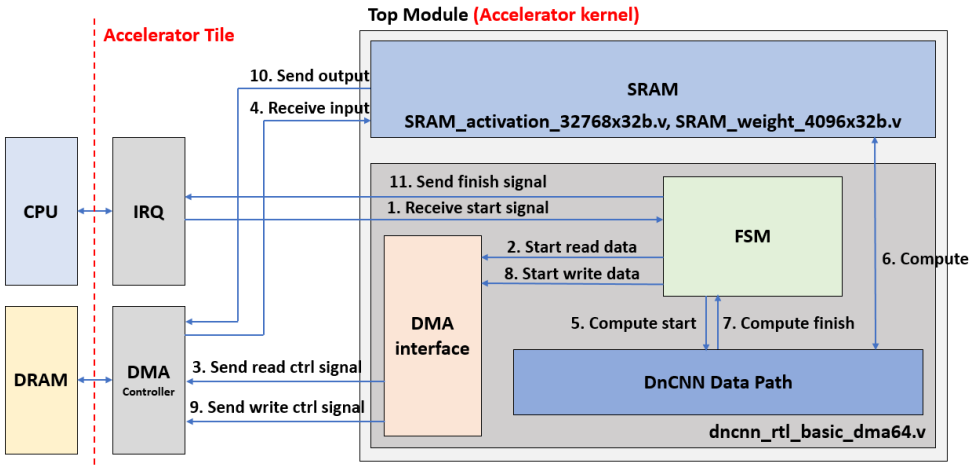
Processing Element

The below figure is the time chart of convolution. We assume filters are loaded into each PE in advance. The first row of input activations are loaded into PE1 in first cycle and start the calculation. In the fourth cycle, PE1 finish calculation and forward the partial sum result to the PE below to accumulate. The second row of input activations are loaded into PE2, PE4 in the second cycle because input activations are reused horizontally. The third row of input activations are loaded into PE3, PE5, PE7 in the third cycle. Later input activations are loaded and calculated in the same way. Since the total cycles of a pipelined architecture is ***(Pipelined stage - 1) + number of instructions***, for one convolution we need (3-1)+3=5 cycles. Therfore, the first convolution result will be generated in the fifth cycle. Then, after fifth cycle, we can get one convolution result at each cycle. If the number of instrtuctions is large enough, the pipelined speedup is about number of pipelined stage, which equals to 3.



Time chart of PE array

C. ESP Block Diagram



D. ESP SRAM

In each sram, we will first group each 2048 addresses then cut each 32 bits into 4x8 bits. After that, we can directly move data in these groups to each BRAM_2048x8 repectively. In total, we use 72 BRAM_2048x8s in this project.

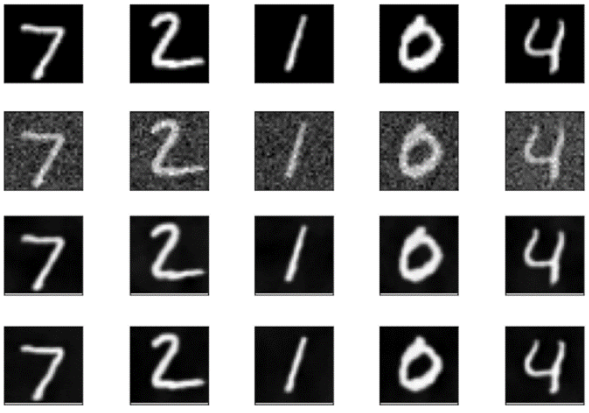
3. 實驗結果

A. Synthesis Result

Item	Description	Unit
Clock period	3.16	ns
Total cell area	108041.000110	um^2
Accelerator cycle count	1147457	cycles
Software cycle count	3506705	cycles
Speedup	3.056	---

B. Visualize

The picture below is the visualization result in this project. From the top, it is the picture of original image, noise-contaminated image, predicted image in software, and predicted image in hardware. The MSE of the predicted image is 0.0069 in software and 0.0382 in hardware. Of course, it is higher in hardware compared to software but somehow it is more similar to the original one. The visualization result shows that our method is successful in the field of image denoising, and we can clearly predict the original image from the noise-contaminated image.



Visualization Result

C. Demo

RTL Simulation Pass

```
Compute start
Compute finished, start validating result...
====================
Image             [PASS]
Conv 1 activation [PASS]
Conv 2 activation [PASS]
Conv 3 activation [PASS]
Conv 4 activation [PASS]
Conv 5 activation [PASS]
Conv 6 activation [PASS]
Conv 7 activation [PASS]
>>> Congratulation! All result are correct
  [RTL simulation]
Clock Period: 3.16 ns,Total cycle count:     1147457 cycles
====================
Simulation finish
Simulation complete via $finish(1) at time 3626291180 PS + 3
./DnCNN_tb.v:212        $finish;
ncsim> exit
```

ESP System-level Simulation Pass

```
# Scanning device tree...
# [probe] sld,dncnn_rtl.0 registered
#         Address   : 0x60010000
#         Interrupt : 6
# ***************** sld,dncnn_rtl.0 ****************
#   memory buffer base-address = 0xa0100b30
#   ptable = 0xa0127c40
#   nchunk = 1
#   --------------------
#   Generate input...
#   -> Non-coherent DMA
#   Start...
#   Done
#   validating...
# ===> Image pass!
# ===> Conv1 pass!
# ===> Conv2 pass!
# ===> Conv3 pass!
# ===> Conv4 pass!
# ===> Conv5 pass!
# ===> Conv6 pass!
# ===> Conv7 pass!
# [PASS] Congratulation! All results are correct
# ** Failure: Program Completed!
#    Time: 42949526400 ps  Iteration: 0  Process: /
```

Setup Time Met

```
clock clk (rise edge)                                   3.16       3.16
clock network delay (ideal)                             0.00       3.16
write_row2_reg[0][31]/CK (DFFRXL)                       0.00       3.16 r
library setup time                                     -0.06       3.10
data required time                                                 3.10
-------------------------------------------------------------------------
data required time                                                 3.10
data arrival time                                                 -3.09
-------------------------------------------------------------------------
slack (MET)                                                        0.01
```

Hold Time Met

```
clock clk (rise edge)                                   0.00       0.00
clock network delay (ideal)                             0.00       0.00
pea/pe00/psum_reg[0]/CK (DFFTRXL)                       0.00       0.00 r
library hold time                                      -0.01      -0.01
data required time                                                -0.01
-------------------------------------------------------------------------
data required time                                                -0.01
data arrival time                                                 -0.17
-------------------------------------------------------------------------
slack (MET)                                                        0.18
```

Area

```
Combinational area:              84831.853592
Buf/Inv area:                     2819.790063
Noncombinational area:           23209.146518
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                108041.000110
Total area:                 undefined
```

4. 結論

We design a denoise CNN model sutible for general image denoising application. With the residual learning method, our model not only exhibit high effectiveness in image denosing tasks, but also reduce training time. Besides, we design a DnCNN accelerator using row stationary method to maximize data reuse avoiding multiple memory access. At last, we integrate our DnCNN accelerator into the ESP platform. By comparing the cycle count of running on software and the accelerator, the result shows that our DnCNN accelerator can efficiently accelerate the DnCNN conputation.

5. 學到的技能或知識

陳奕帆:

I learned the deep learning model computation and different dataflows to design an accelerator in the class. In the final project, I implemented the row stationary method to design the accelerator. There are many things need to consider. Such as how to deal with different kernel size, different input/output feature map size, different channel amount. How to design the dataflow to maximize data reuse when encounter different types of convolution? I learned what need to take into considerations to design a high performance yet energy efficient accelerator. Besides, the software/ hardware cooperation is very important to design a practival product. At last, I learned how to cooperate and communicate with others to finish a project. I will use the knowledge I learned from this class as future research motivation.

小山翼:

In this project, I first studied the details of Quantization Aware Training (QAT) since we only practice the Post Training Quantization (PTQ) in this class except the knowledge. While constructing the QAT model, I found that there are many techniques to complete the task. I spent a lot of time judging whether the method fits our model or not. In ESP, I learned how to use two Finite State Machines (FSMs) to do the whole integration since I only use one FSM in midterm project and the integrating process is a little bit inefficient. And this is my first time to do such a large coding project with others, I learned how to do work division in programming, cooperate, and communicate with each other. I hope that I can make flexible use of the knowledge learned in this class.

6. 組員分工與開發時程

A. Task description of each member

Group member	Task description
小山翼	DnCNN Quantization, ESP Integration
陳奕帆	DnCNN model, DnCNN Accelerator

B. Devolped time

Item	Devolped time(days)
DnCNN model	2
DnCNN Quantization	7
DnCNN Accelerator	14
ESP Integration	8

7. 參考資料

[1] Image Denoising | Vision and Image Processing Lab, https://uwaterloo.ca/vision-image-processing-lab/research-demos/image-denoising

[2] Inside Quantization Aware Training | by Vaibhav Nandwani, nside Quantization Aware Training | by Vaibhav Nandwani, https://towardsdatascience.com/inside-quantization-aware-training-4f91c8837ead

[3] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, Lei Zhang, “Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising”, https://arxiv.org/abs/1608.03981

[4] Practical Quantization in PyTorch by Suraj Subramanian, Mark Saroufim, Jerry Zhang, https://pytorch.org/blog/quantization-in-practice/

[5] Y. -H. Chen, T. Krishna, J. S. Emer and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127-138, Jan. 2017, doi: 10.1109/JSSC.2016.2616357.