ECSE 682: Topics in Computers and Circuits - Project

Due Date: Dec. 18, 2017

Fall 2017 - McGill University - Electrical & Computer Engineering

Prof. Warren Gross

Intorduction

In this exercise, you will experiment with a convolutional neural network (CNN) to solve a classification task on CIFAR10 dataset using Ordinary People Accelerating Learning (OPAL) tool. CNNs consist of multiple convolutional layers followed by fully-connected layers. In CNNs, convolutional layers extract high level abstraction and features of raw data. The connectivity of the convolutional layers follows a pattern inspired by the organization of the animal visual cortex, that can mathematically be described by a convolution operation. A set of weights, i.e. a filter, is shared among all neurons in these layers. Each convolutional layer provides as output the activation map, that is a higher level of abstraction with respect to the input data. The main computation kernel of a convolutional layer processes high-dimensional convolutions, summarized in Fig. 1.

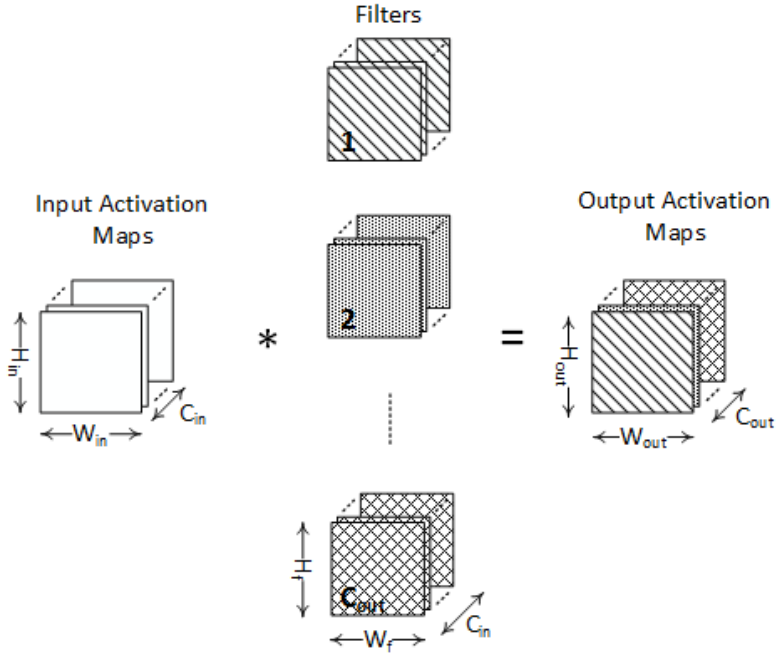

Figure 1: *The computations of a convolutional layer.*

Contrary to the fully-connected layers, the convolutional layers consist of neurons (pixels of the activation maps) arranged in 3 dimensions: height H, width W and channel C. Each neuron performs multiplications and accumulations and is implemented using MAC units in

convolutional layers. Therefore, each convolutional layer transforms 3D input activation maps to 3D output activation maps. This transformation is a result of the convolution between input activation maps and 4D filters (i.e. a set of C_{out} 3D filters). Each set of 3D filters is convolved by the 3D input activation maps to give a single 2D $H_{out} \times W_{out}$ plane of the output activation maps. More precisely, 3D convolution is a summation of multiple plane-wise 2D convolutions. Moreover, a 1D bias is added to the 3D convolution results.

Name of Parameter	Description
H_{in}/W_{in}	Height/Width of the input activation maps
H_{out}/W_{out}	Height/Width of the output activation maps
H_f/W_f	Height/Width of the filter plane
C_{in}/C_{out}	No. of channels of the input/output activation maps
S	Stride number

Table 1: *Convolutional Layer Computation Parameters.*

Fig. 1 illustrates the computations performed in a convolutional layer, while the parameters used in such calculations are given in Table 1. Each layer performs the following operations:

$$Y(z,t,q) = B(q) + \sum_{k=1}^{C_{in}} \sum_{j=1}^{H_f} \sum_{i=1}^{W_f} X(zS+j, tS+i, k) \times W(j,i,k,q),$$

$$H_{out} = (H_{in} - H_f + S)/S,$$

$$W_{out} = (W_{in} - W_f + S)/S, \quad (1)$$

where X, Y, W and B are the input activation maps, the output activation maps, the filters and the bias matrices, respectively, and $1 \leq z \leq H_{out}$, $1 \leq t \leq W_{out}$ and $1 \leq q \leq C_{out}$. Stride S is also defined as the number of pixels of the activation maps of which the filter is shifted after each convolution. Read the reference below for more information on convolutional processes.

V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," arXiv e-prints abs/1603.07285 (2016).

Part 1: Training Neural Networks

Use OPAL to find a CNN that yields less than 25% misclassification rate on CIFAR10. Note that a common choice of activation function is the ReLU function (i.e., *ReLU(x) = max(0,x)*) which is more hardware-friendly than the other activation functions such as *sigmoid* and *tanh* functions. Therefore, using the ReLU function is recommended although you are not limited to it. Report the network configurations, in particular number of hidden layers and the parameters listed in Table 1.

Part 2: Fixed-Point Model

Write a fixed-point program of your own choice that implements the inference engine (i.e., feed-forward computations) of the designed network above. Note that the floating point weights can be found as a numpy array (i.e., the file with .npy extension) in *./results* directory. Find the minimum possible word-length that performs the computations with the misclassification rate below 25%.

Part 3: Hardware Implementation

The computational complexity of CNNs is dominated by the convolutional layers, whereas the fully-connected layers contain the majority of network parameters: fully-connected layers are thus easy to implement in hardware. In fact, the main core of convolutional layers has a higher degree of computational complexity, dominated by the numerous multiplications/additions. As a result, using a hardware accelerator is a common trend among researchers to accommodate convolutional computations. In this oart of the project, you will implement a hardware accelerator for the convolutional layers (i.e., you do not have to implement pooling or fully-connected layers).

a. Design a hardware accelerator that can process any of the convolutional layers in your network obtained in part 1.

b. Derive a systolic architecture using scheduling and projection vectors of your own choice.

c. Implement the convolutional accelerator that you found in part b in VHDL/Verilog targeting a FPGA by following the steps below.

    1. Depending on your architecture, load the entire or a part of weights, biases and input pixels required for the computations of the first convolutional layer of your network into BRAMs available on the FPGA board. Assume that the weights, biases and input/output activation maps of each convolutional layer are stored in large SRAM memories, referred to as off-chip memories, in your testbench code. Note that all the input/output data including both weights, input and output pixels are read serially (i.e. a single input at each clock cycle).
    2. Perform the computations of the first layer.
    3. Store the output activation maps of the first convolutional layer in the off-chip memory.
    4. Test the remaining convolutional layers of your network using the steps above.

d. Characterize and fully-verify your design by reporting number of MACs, total latency of the convolutional computations, throughout (frame per second), number of memory accesses to the off-chip memory (MByte), performance in terms of operations per second (Ops) considering each MAC as two operations, and complete FPGA resource utilization.

Part 4: Competition

Repeat parts 1, 2 and 3 to optimize your convolutional accelerator achieving the lowest possible memory accesses and the highest throughput which can fit in Cyclone V E (5CEFA2U19I7N) FPGA. The goal of this question is to find the best trade-off between memory accesses and

throughput for a hardware implementation of CNNs. Note that the grade of this part will be assigned according to how you do in the competiotn.
