209AS Lab1

Due date: May 3rd midnight

Ver 1.3

Update log:

Ver 1.1: Changed max pooling in part 1 (b) and (c) to average pooling to be consistent with part 1 (a).

Ver 1.2: Fixed an error in the part 2 description regarding total dot products per cycle (128 -> 16), as well as the description regarding to total weight vector need to be processed per cycle. Also added an example for latency numbers of part 2.

Ver 1.3: Updated due date. Changed the report page limit from 6 to 8.

Refined the description of Part 2 problem (a) to avoid confusion. Added code output format for Part 2.

Part 1

This is part 1 of lab1. In this part, you will implement a custom network called Lab1Net, following the description below, and train and evaluate the network using CIFAR-10 dataset. You then need to create a custom conv2d wrapper to introduce some noise to activations. The main goal of this part is to let you familiarize yourself with common PyTorch syntax as well as how to custom neural network layers (quite important).

Network definition

1. The network consists of three convolutional layers followed by a fully connected layer.

2. All convolutional layers use 3x3 filters with zero-padding of 1 and no bias. The zero padding are added to make sure the activation size is the same after the convolution operation.

- The first convolutional layer has 3 input channels and 32 output channels.

- The second convolutional layer has 32 input and output channels.

- The third convolutional layer has 32 input channels and 64 output channels.

3. Each convolutional layer is followed by:

- Average pooling with a pool size of 2.

- Batch normalization

- ReLU activation.

4. After the third convolutional layer and its subsequent operations, flatten the output to a 1D tensor.

5. The fully connected layer takes the flattened output and maps it to 10 output classes, without bias.

6. Apply batch normalization to the output of the fully connected layer.

The forward pass of the network should follow this sequence of operations:

1. Pass the input (activation) through the first convolutional layer, average pooling, batch normalization, and ReLU activation.

2. Repeat step 1 for the second convolutional layer.

3. Repeat step 1 for the third convolution layer.

4. Flatten the output and pass it through the fully connected layer.

5. Apply batch normalization to the output of the fully connected layer.

6. Return the final output.

Problems

(a) [5 pts] Train and evaluate the provided network on CIFAR-10 dataset. You should configure the optimizer, number of epochs (less than 100) and other hyperparameters properly to ensure good accuracy. There is no hard accuracy target for this problem and there will no competition in this problem, but you should try a few different options and observe how they may (or may not) impact the accuracy. Also, store the trained weights as a .pth file using torch.save, as you need to use this weight later. You can connect colab to drive and store the weights to your google drive directly, or you can just manually download the .pth file.
In your report, show how the forward function of the model class is implemented, and report your achieved accuracy along with hyperparameters such as optimizer, learning rate, number of epochs, etc.

(b) [5 pts] In the original network definition, average pooling is used to reduce the activation size. However, there is another way to achieve the same activation size reduction effect, which is strided convolution. Convolution strides refer to the number of pixels by which a convolutional filter moves across an image during the convolution operation.
In this problem, you need to remove all average pool layers in the network, and

adjust the stride of each conv layer to achieve the same effect. The input activation size of each layer needs to be the same after the change. Train and evaluate again with this configuration and compare with the results you obtained from (a). Report your results, how the results compared to (a), and try to explain the difference (if any). Save your trained weights.

(c) [5 pts] For this problem, you need to create your own convolution wrapper class Conv2d_custom to replace nn.Conv2d that you used in previous problems. You can still use PyTorch's F.Conv2d's for the actual convolution. The source code of nn.Conv2d (https://github.com/pytorch/pytorch/blob/main/torch/nn/modules/conv.py) is a good starting point for your own wrapper class. However, make sure you import and include necessary classes and functions.
The task of this project is to introduce some random noise to the inputs of each convolution layer, and you need to implement it using your Conv2d_custom class.

The network architecture should be the same as problem (a), which uses average pooing and stride = 1. For each convolution layer, add a gaussian noise to the input activation. The mean of the noise is 0 and std is 0.1 . Now, load the weights you stored from (a), and run inference without training. What do you observe?
Then retrain the model with input noise for 10 epochs, and evaluate the accuracy again. Does the accuracy improved compared to direct inference?
In report, write about how do you add noise to the input and report the accuracy before and after retraining. Save your trained weights.

You should implement and run the problems using google colab, unless you'd prefer to run them on your own machine. In colab, you can store the Jupiter notebook (ipynb) file into normal .py python file using Download option from File tab.

Submission instructions (read carefully)

You need to submit the codes as well as a report in the form of slides. The report needs to be submitted to Gradescope while the codes and weights needs to be submitted to the seasnet server *eeapps.seas.ucla.edu* or *linuxapps.seas.ucla.edu.*

If you are working in teams, then submit using team submission on Gradescope, and each team only needs to submit once. Do not submit twice. The same applies to code submission, each team only need to submit only once.

The report should be **6-8 pages slides (maximum 8 page) for part 1 and part 2 combined** (but in PDF format), summarizing your approach and results.

Regarding the code. **You need to submit three .py python files to the server for this part, one for each problem, along with the corresponding weights in .pth format.** The filenames for codes should be lab1a.py, lab1b.py, nb lab1c.py respectively, and for weights should be lab1a.pth, lab1b.pth, and lab1c.pth.

Detailed code submission instruction

1. Create a directory in your home directory on the server named as follows: UID_Lastname_Firstname_Lab1p1
2. Upload all required codes and weights into this directory
3. Compress and archive this directory using tar to have a single tarball named: UID_Lastname_Firstname_Lab1p1_pinXXXX.tar.gz (Make up a 4-digit numeric PIN of your choice to substitute for XXXX to avoid others guessing the filename)
4. **Important: before submitting, make sure all the files in the tarball, as well as the tarball itself, have full read and execute permissions to groups and others**, otherwise your files cannot be graded.
5. Submit tarball by copying it to '/w/class.1/ee/ee209w/ee209wt2/submission/project1/'
6. **Late submissions will not be accepted,** the write permission of the submission directory will be removed after deadline

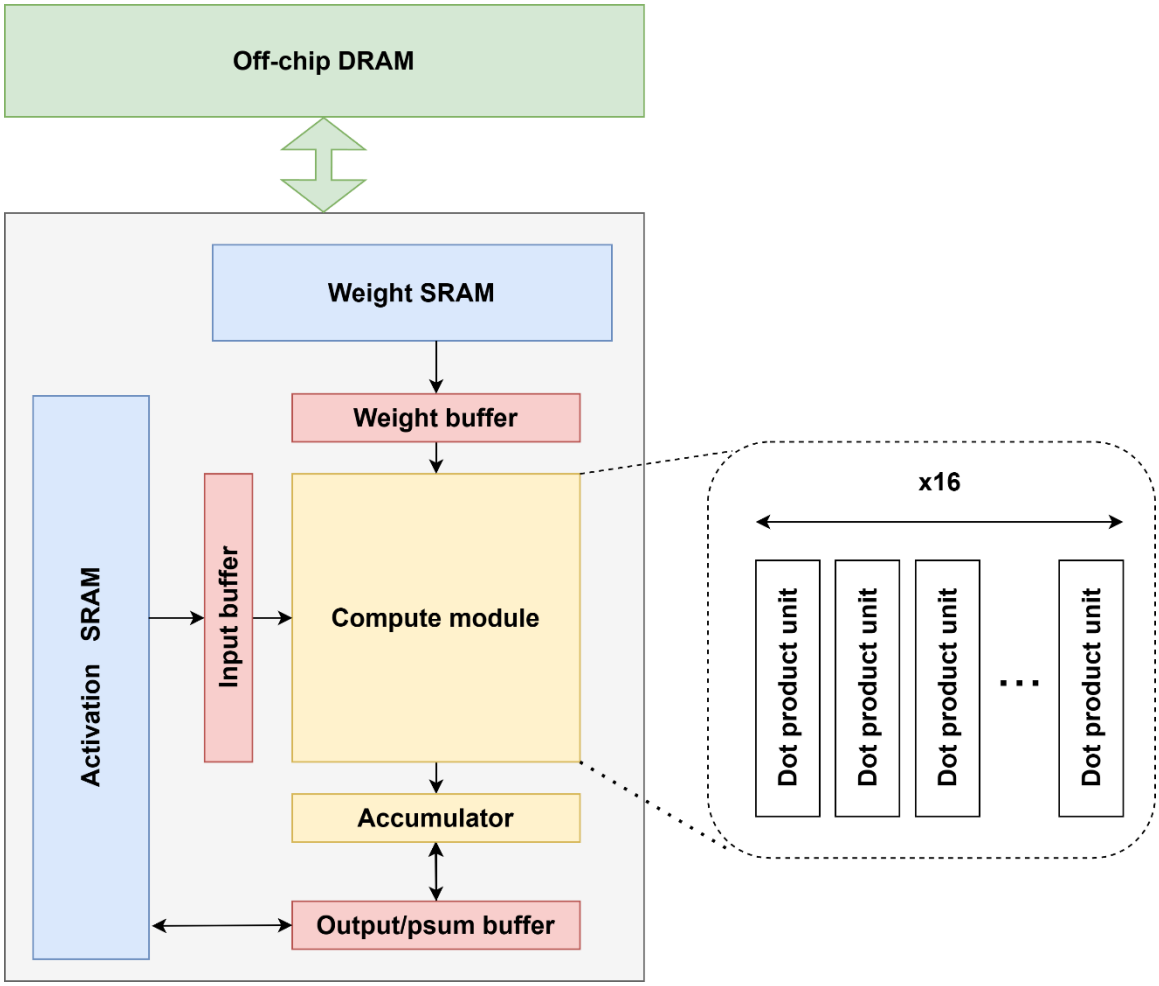
Submission example step-by-step

```
$ cd <YOUR_LAB1_WORK_DIRECTORY>
$ mkdir 666666_Li_Shurui_lab1p1
Upload all the files into this folder, you can either use
MobaXterm's SSH browser or using scp
$ chmod -R go+rx 666666_Li_Shurui_lab1p1
$ tar -czf 666666_Li_Shurui_lab1p1_pin7777.tar.gz
666666_Li_Shurui_lab1p1
$ chmod go+tx 666666_Li_Shurui_lab1p1_pin7777.tar.gz
$ cp 666666_Li_Shurui_lab1p1_pin7777.tar.gz
/w/class.1/ee/ee209w/ee209wt2/submission/project1/
```

Part 2

In this problem, you will analyze the energy consumption, latency, and throughput of a convolutional neural network (CNN) accelerator system using different dataflows (WS, IS, and OS) and batch sizes. You will also explore the impact of pipelining the system on latency and throughput. You need to write a simple parameterized model/simulator for this problem using Python, don't make hand calculations except for verifying your simulator.

The compute module contains 16 dot product units each with a size of 128, and the entire module can perform 16 dot products in parallel in one cycle, that is 2048 MAC (multiply and accumulate) per cycle. Each cycle, the compute module takes a single input vector with length 128 from the input buffer and broadcast the input vector to all 16 dot product units, while each dot product unit takes one unique weight vector with length 128 from weight buffer (total 16 weight vectors each with size 128). For a convolution layer, since the same input vector is broadcasted to all dot product units, naturally each dot product unit will process a unique convolution filter.



As show in the figure, the entire system consists of the compute module (including the input, weight, and output/psum buffer), a weight SRAM, a unified activation SRAM (for both input and output activations). The system also contains a DRAM which stores all the weights.

To simplify the problem as much as possible, many assumptions are made.

Assumptions and notes

- IM2COL will be used for convolution layers. You can assume the inputs are somehow automatically transformed using IM2COL **within** the activation memory, which essentially **makes the input matrix N times larger**, where N is the filter size. This assumption applies to all dataflow including IS dataflow. You can ignore the energy and latency overhead of the IM2COL transformation.
- You can ignore the energy spent on the input, weight, and output buffers.
- The output buffer includes an accumulation unit, which can accumulate outputs from consecutive cycles in OS dataflow before writing into activation SRAM. You can ignore the energy of this accumulation unit as well.
- The weight SRAM is large enough to hold the full weights of the entire neural network, but they need to be loaded from DRAM first, before the actual processing starts.

- The activation SRAM is large enough to hold all the input/output activations during execution for all batch sizes considered in this problem.
- Use 'valid' mode for the convolutions (no padding), which means the output size will be slightly smaller than the input size. E.g., 32x32 input convolves with 3*3 filter, output size is 30x30.
- For the case where number of filters is less than the number of dot product units (if there are such cases), then the dot product units will be underutilized.
- When the dot product size is less than the dot product unit size (128), inputs and weights will be zero padded automatically, and you should not count the energy for the zero padding. E.g., if only 10 values are loaded in a dot product unit, only count access energy for those 10 values, not 128. This won't affect the dot product unit energy.
- For batch size > 1, all inputs in the batch need to be processed for the current layer before starting the next layer.
- For all calculations, we only care about the convolution layers, ignore everything else.

System specs

Activation bitwidth: 8
Weight bitwidth: 8
DRAM access energy: 4pJ/bit
SRAM access energy: 0.1pJ/bit
Number of dot product units: 16
Dot product unit size: 128
Single dot product unit energy per cycle: 20pJ
Clock period: 1 ns

All latency number here are number of cycles required to do the operation. And for memory load it means the latency to read all required data from the SRAM to buffer, same for memory write. For example, it takes 5 cycles to read 16 weight vectors (each with 128 elements) from weight SRAM to weight buffer.
Computation latency means the latency required to perform one dot product operation for all 16 dot product engines in parallel. DRAM latency means the latency required to load weights from all layers into weight SRAM.

DRAM access latency: 7000
Weight SRAM load latency: 5
Activation SRAM load latency: 3
Activation SRAM write latency: 3
Compute latency: 2

Neural network and inputs for evaluation

Input shape is 3x32x32 (3 is number of input channel)
Neural network definition:
Format: Conv [#F, #C, X, Y] -> number of filters, number of channels, filter width, filter height

1. Conv [32,3,3,3]

2. Conv [64,32,3,3]
3. Maxpool2d (reduce both width and height of activation by 2)
4. Conv [128,64,3,3]
5. Conv [256, 128, 3,3]
6. Conv [512, 256, 3, 3]

Notes on Neural network dataflow

In convolutional neural network (CNN) accelerators, dataflows determine the order and manner in which data (activations and weights) are brought to the compute units, processed, and stored back to memory. In this project, we will focus on three common dataflows: Weight Stationary (WS), Input Stationary (IS), and Output Stationary (OS).

Weight Stationary (WS) Dataflow: In the WS dataflow, the weights are kept stationary in the local memory (e.g., weight buffer) of the processing elements (PEs, in this case dot product units) for as long as possible. The activations are streamed into the PEs, and the partial sums are accumulated and moved out.

Input Stationary (IS) Dataflow: In the IS dataflow, the input activations are kept stationary in the local memory of the PEs, while the weights are streamed in. The partial sums are accumulated and moved out of the PEs.

Output Stationary (OS) Dataflow: In the OS dataflow, the output activations (partial sums) are kept stationary in the local memory of the PEs. The weights and input activations are streamed into the PEs, and the partial sums are accumulated locally without the need to access partial sums from memory.

Problems:

(a) [15 pts]

For the convolution neural network, your task is to generate the total energy spent on accessing DRAM, weight SRAM, activation SRAM, and the compute module for different dataflows, **when processing a single batch**.
You need to generate results for two batch sizes, **batch size = 1**, and **batch size = 256**.
Use Joule (J) to report your energy results.
Provide a detailed analysis and comparison of the energy consumption for each listed dataflow.

1. WS dataflow (Weight stationary)
2. IS dataflow (Input stationary)
3. OS dataflow (Output stationary)

(b) [10 pts]

Generate the latency and average throughput for the cases where batch size is **1 and 256**. **Use WS dataflow for this problem**. Use seconds for the unit of latency (the time to complete a single batch) and use FPS (frames/inputs per second) for the unit of average throughput.

Additional assumptions for this problem:

- The system is not pipelined, that means memory read, computation, and memory write must happen sequentially, they cannot be overlapped. The exception is reading from weight SRAM and reading from activation SRAM (to input buffer and/or output buffer), they can happen in parallel as they do not share resources and has no dependency
- Don't try to optimize the hardware or be 'smart', just follow the assumptions.

(c) [5 pts]

What if we pipeline the system? Repeat the calculations of (b) for the pipelined case.

Additional assumptions for this problem:

- You are only allowed to pipeline the system into 3 stages – memory read, computation, and memory write. Don't pipeline the system further.
- You can ignore the overhead of the additional hardware required to implement such pipeline.

In your report, you should list the results for each problem, and explain how your simulator/model is implemented to obtain these results. Also, for each problem, compare the results between different setups (such as dataflow and batch size), and explain your observations. **Make sure to not give results without any explanation, as you will lose all points if your results are not correct.**

Submission instruction:

As mentioned in part 1**, you need to submit a <= 8-page report for part 1 and part 2 combined to gradescope, summarizing your results and approach.**
Regarding the codes, assuming you are using python for your simulator, you should submit 3 python scripts, one for each subproblem. Your scripts should be implemented that by running them the required results (and only required results) are printed out. Do not hardcode the results and print them out, all results should be dynamically generated by your simulator scripts. The name of the require python scripts are lab1p2a.py, lab1p2b.py, and lab2p2c.py, respectively.

Code print format:
(a): 6 lines, each line with format *"dataflow = XX, batch size = XX, energy = XX J*".
(b): 2 lines, each line with format *"batch size = XX, latency = XX s, average FPS = XX"*.
(c): Same as (b).

Detailed code submission instruction

7. Create a directory in your home directory on the server named as follows: UID_Lastname_Firstname_Lab1p2
8. Upload all required codes into this directory

9. Compress and archive this directory using tar to have a single tarball named: UID_Lastname_Firstname_Lab1p2_pinXXXX.tar.gz (Reuse your PIN for part 1 of this project)
10. **Important: before submitting, make sure all the files in the tarball, as well as the tarball itself, have full read, and execute permissions to groups and others**, otherwise your files cannot be graded
11. Submit tarball by copying it to '/w/class.1/ee/ee209w/ee209wt2/submission/project1/'
12. **Late submissions will not be accepted,** the write permission of the submission directory will be removed after deadline

Please refer to the description of part1 for the detailed submission example.