Assignment № 2

Deadline: 11:55pm Nov. 1, 2018

Learning Outcomes

- Apply CUDA programming to one of the most popular field: Deep Learning
- Execute deep learning inference with your own program
- Deep understanding of shared memory
- Get better preparation for the term project

What is CNN

From [2]: (recommended reading)

In recent years, deep learning has been used extensively in a wide range of fields. In deep learning, Convolutional Neural Networks are found to give the most accurate results in solving real world problems. CNN is used in computer vision, mainly in face recognition, scene labelling, image classification, action recognition, human pose estimation and document analysis.

In the remaining parts of this page, we will look into the implementation of CNN and the explanation of the provided codes.

Implementation of CNN

This assignment uses a CNN that is trained for speed sign detection and recognition. For example, the picture below shows the detection and recognition of three speed signs (in red box).

The detection is performed in multiple layers, each consists of multiple feature maps. Below is an example of the CNN with a small input image of 32x32 pixels. In the assignment, we will use a large input image of 1280x720 pixels, and an optimized CNN that merges the convolution layer and subsampling layer.

There is a great video for convolutional neural network: link

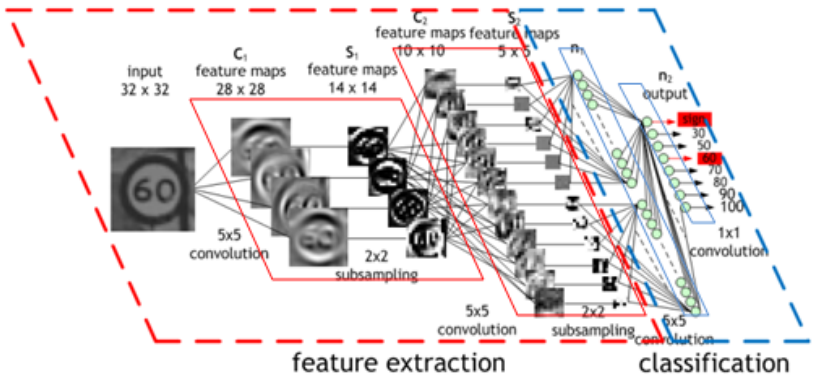
The convolution can be written with a few variables and a sigmoid function:

1. The bias (b).
2. The input (x) and weight (v), of size K (or K*K for 2D case).
3. The sigmoid function (ϕ).

The convolution can be illustrated in this link

The sigmoid function is described by the equation:

$$\Phi = \frac{1}{1 + exp(-x)}$$



For a two dimensional case, the convolution can be described by the equation:

$$y[m,n] = \Phi(p) = \Phi(b + \Sigma_{k=0}^{K-1}\Sigma_{l=0}^{K-1} v[k,l]x[m+k,n+l])$$

Example of the First Convolution Layer

The first layer has one input map (the input image) of size 1280x720 and 6 output maps, 638x358 each. The convolution window has a size of 6x6. The layer is implemented as nested loops as shown below.

```
1
2 for(r=0; r<6; r++){ //loop over 6 feature maps
3     for(m=0; m<358; m++){ // loop over 358 rows
4         for(n=0; n<638; n++){ // loop over 638 columns
5             y[r*358*638+m*638+n]=bias[r]; // {Step 1: load bias
6             for(k=0; k<6; k++){ //loop over 6x6 window
7                 for(l=0; l<6; l++){ // loop over 6x6 window
8                     y[r*358*638+m*638+n] += in_layer[(m*2+k)*1280+n*2+l] *↩
```



```
                        weight[r*36+k*6+l]; // Step 2: convolution
            }
        }
        out_layer[r*358*638+m*638+n]=(unsigned char)(255.999f/(1+expf(-y[r*358*638+m*638+n]/256))); // Step 3: sigmoid function
    }
  }
}
```

To compute each output element on the feature map, three major steps are performed:

1. Load the bias.
2. Perform convolution over a window.
3. Apply the sigmoid function to the result.

Other layers have similar structures. So you only need to implement the first layer.

Your Task: The GPU Implementation of the First Convolution Layer

The file "**cuda_functions.cu**" consists of the empty CUDA codes for the first convolution layer. You are supported to:

1. finish the three kernel codes in "**cuda_functions.cu**" **WITHOUT** shared memory. You need to use the block size specified in the CUDA file (in comments) (20%)
   In "**layer1_init_bias**", you do Step 1 in the above CPU code with GPU. With block size: (16, 16, z) (choose your z dimension)
   In "**layer1_feature_maps**", you do Step 2 in the above CPU code with GPU. With block size: (8, 8, z) (choose your z dimension)
   In "**layer1_sigmoid**", you first do Step 3 in the above CPU code with GPU. With block size: (14, 14, z) (choose your z dimension)

   In the code comments, there is detailed explanation of the logic of this program.

   You may see that what you need to do is just converting the above CPU code to a GPU one.

2. use shared memory to cache the input (i.e. the image). Your codes should have reasonable performance.
   The baseline takes 4.4 ms for the convolution. You will get full marks in this part if your code takes $(0, 5]$ms, $\frac{3}{4}$% marks if your code takes $(5, 6]$ms, 0 marks if your code takes more than 6 ms. You need to use the block size specified in the CUDA file (25%)

   Hints will be given in next tutorial.

3. you can NOT use hard-coded number. You need to define them as macro at the beginning of the "**cuda_functions.cu**" with reasonable name. Meanwhile, you need to give explanations of each number as comments. Otherwise, you get 0 mark from this part. (30%)

   For example, if you want to declare a variable "**int figure_size = 1024 * 768;**", where 1024 is the width and 768 is the height. Then, you need to declare 1024 and 768 be macros with explanations(see following).

```
1        #define height 1024 //picture height
2        #define width 768 //picture width
3        ...
4        int figure_size = 1024 * 768; //Wrong! This is hard coded number, why 1024 and why 768?
5        int figure_size = height * width; //size of the figure (correct one, we know that figure_size is computed by height times width)
6        ...
```

4. write a tiny report to explain your solution and compare the execution time between 1 and 2. (15%)

5. error check, memory free, code style. (10%)

You must not change the declarations and signatures of the kernel functions, otherwise, you get 0 mark of this assignment!

Submission

You need to submit all the code files and your report as a .zip file.

Reference

[1] https://developer.nvidia.com/cuda-example

[2] http://ijcsit.com/docs/Volume%207/vol7issue5/ijcsit20160705014.pdf

[3] https://sites.google.com/site/5kk73gpu2013/assignment/cnn

[4] http://machinelearninguru.com/computer_vision/basics/convolution/convolution_layer.html

[5] https://www.youtube.com/watch?v=jajksuQW4mc

