ECE 5755 Modern Computer Systems and Architecture, Fall 2023
Assignment 1: Implementing an ML Kernel

1. Introduction

This lab assignment is an introduction to the fundamental techniques we'll be using throughout the semester for performance analysis. You will be implementing your own machine learning library in C for the forward pass of a small convolutional neural network. This lab will serve as a base for the rest of the labs where you will be applying different optimizations to your kernel.

2. Materials

lab1.zip contains all of the files you need to build the project:

- Makefile: a makefile for you to easily (1) compile source code into an executable and (2) execute the program to generate results
- kernel: a basic and **incomplete** ML kernel that you are tasked with implementing
    - kernel.h: header file with useful includes and information
    - conv.c: where you will implement the **convolution** function
    - conv.h: header file for conv.c with useful includes and information
    - functional.c: where you will implement the activation functions **relu** and **softmax**
    - functional.h: header file for functional.c with useful includes and information
    - linear.c: where you will implement the **linear** function for a fully-connected layer
    - linear.h: header file for linear.c
    - matrix_ops.c: where you will implement the function **matmul**
    - matrix_ops.h: header file for matmul.c
- tests: incomplete test suite where you will complete any incomplete tests
- utils: utility functions for driver code

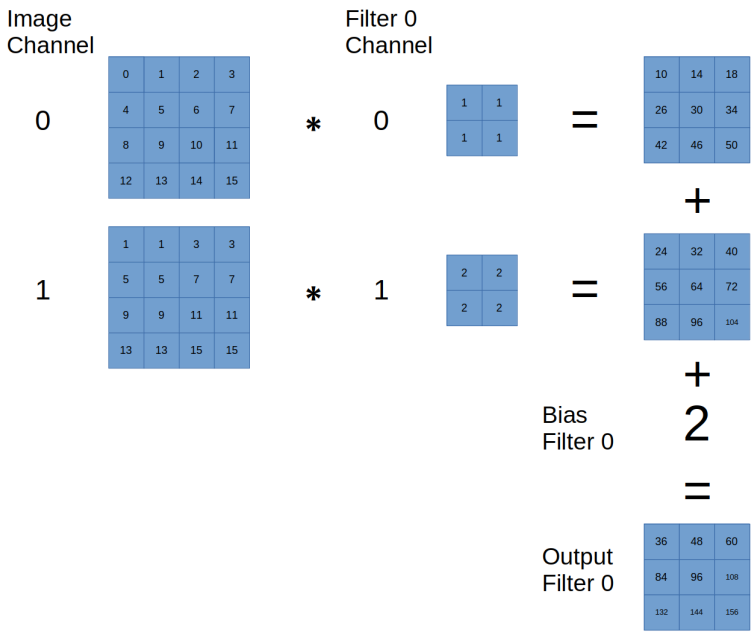
Copy the lab1 files to the course server as you did in lab 0.

3. CNN Forward Pass Algorithm/Architecture

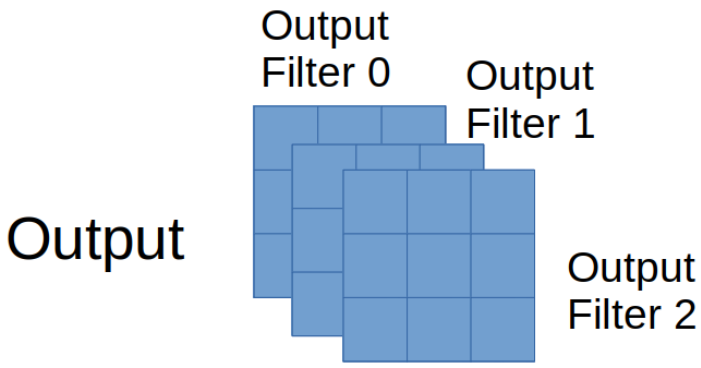
Read this first before proceeding if you are not familiar with CNNs: https://www.ibm.com/topics/convolutional-neural-networks. You will not be implementing the forward pass algorithm in this lab, but the algorithm is covered briefly here for your reference. The input image is a three-dimensional array. The first dimension represents the channel of the image. For example, an RBG image will have three channels, one for each of red, green and blue. The second dimension represents the row and the third dimension represents the column. For example, image[0][4][2] would be the 0th channel value for the pixel at row 4, column 2.

1. Convolution layer (stride of 1, no padding)

The kernel is a four dimensional array with one or more filters. The first dimension of the kernel array represents which filter. The second dimension of the kernel array represents the channel. The third dimension represents the row and the fourth dimension represents the column. For example, kernel[3][1][2][1] is the weight (or value) at the (2, 1) position for the 3rd filter's 1st channel. Note that the results are summed across all channels for the output so the output will be a three-dimensional array with the first dimension representing depth (i.e., which filter within the kernel), the second dimension representing row, and the third dimension representing column. For example, if there are three filters, then the depth will be three and the first dimension of the output will range from 0 to 2.



Convolution example for one filter



Example convolution output for a kernel with three filters

The size calculations and memory allocation for the convolution have been included for your convenience. Each filter has a bias term that must also be applied across all of that filter's output values. For example, biasData[1] must be added to convOutput[1][i][j] for all i, j.

Finally, the ReLU activation function (see https://en.wikipedia.org/wiki/Rectifier_(neural_networks)) is applied to all values of the output array. **You will need to implement the ReLU activation function in relu().**

You will need to implement the entire convolution layer (including addition of the bias term and application of ReLU) in convolution().

In future labs, we will also provide you with a version of convolution() that uses matrix-matrix multiplication. In this lab, as preparation for the future labs, **you will need to implement matrix-matrix multiplication in matmul().**

2. Flatten

Flatten turns the multi-dimensional output of the convolutional layer into a one-dimensional vector so it can be used as an input into a fully-connected layer.

3. Fully connected layers

The flattened vector is fed into two fully connected/linear layers as the input. The ReLU activation function is applied between the two fully connected/linear layers. **You will need to implement the fully connected/linear layer in linear().** linear() should only perform vector-matrix multiplication and not include the ReLU activation function.

4. Softmax classification

A softmax activation function is used to convert the output vector from the fully connected layers into a discrete probability distribution (see https://en.wikipedia.org/wiki/Softmax_function). Finally, the class with the highest probability is chosen as the predicted class to classify the input image. **You will need to implement the softmax function in softmax().** softmax() should only implement the softmax function and not the prediction function.

5. Testing and Build

A partial test suite has been provided for you under the tests folder. The test suite uses the Unity testing tool: http://www.throwtheswitch.org/unity. You need to complete the testing functions in the following files under the tests folder:

- test_conv.c
- test_linear.c
- test_matrix_ops.c

Look at the provided completed test functions within the tests folder for examples of how to write tests for your functions. Note that all_tests.c is the driver program that will execute each of the tests.

If you are having issues with passing the tests, look through the test code to understand what the tests are doing and what the expected results are. To build and run the tests:

```
$ make all_tests
```

6. Profiling

To make your code faster, you need to understand what the bottlenecks are for each function. Using the test suite, collect Top-down analysis data for each of the following functions:

- convolution()
- relu()
- linear()
- matmul()
- softmax()

What are the architectural bottlenecks for each? Make sure you are testing with a variety of input sizes (at least 3 inputs per function). Note that for each test, there is a lot of setup, verification, and cleanup code. How can you minimize the impact of the code outside of the function you are profiling? *Hint: repetition*

5. Report

The report will be **3-pages max**. Include in your report a horizontal segmented bar chart for the Top-down analysis profile of each of the following:

- convolution()
- relu()
- linear()
- matmul()
- softmax()

Explain what you did to profile each of the above. Discuss any observations made about the profiling data. Based on your Top-down analysis data, where are the main bottlenecks for each of profiling test cases and why?

Explain your implementation of each of the following functions:

- convolution()
- relu()
- linear()
- matmul()
- softmax()

Explain the tests you implemented:

- test_conv.c
- test_linear.c
- test_matrix_ops.c

Discuss any issues you had during the lab and your debugging process.

Discuss any ideas you have about improving the performance of the code with reference to the Top-down analysis data you collected.

6. Deliverables

Please submit your assignment on Canvas by **Tuesday, Sept. 26, 11:59 pm**. You are expected to submit your code and scripts in a single file named **lab1.zip** and your report in a file named **lab1.pdf**.

The lab will be marked out of a **total of 100 marks.**

- **30 marks** for correctness of kernel functions (as tested by our internal grading tests)
    - 10 mark for convolution()
    - 5 marks for relu()
    - 5 marks for linear
    - 5 mark for matmul()
    - 5 marks for softmax()
- **20 marks** for unit tests you implemented
    - 10 mark for test_conv.c
    - 5 marks for test_linear.c
    - 5 marks for test_matrix_ops.c
- **50 marks** for report
    - 25 marks for Top-down analysis data
        - 5 marks for presenting data as horizontal segmented bar chart
        - 10 mark for each including analysis data for all required functions
        - 10 mark for having at least 3 test inputs per function
    - 25 marks for discussion (see prompts in 5. Report)