Laboratory 2

(Due date: Oct. 4th)

Objectives

- Compile and execute C/C++ code in Ubuntu 12.04.4 using the Terasic DE2i-150 Development Kit.
- Perform image convolution on an image read as a binary file.
- Learn about image displaying considerations.
- Using a Makefile to compile C programs.
- Implement a Neural Network Layer using functors in C++.

Terasic DE2i-150 Development Kit

Documentation

- Refer to the board website or the Tutorial: Embedded Intel for User Manuals and Guides.

Tutorials

- Refer to the Tutorial: High-Performance Embedded Programming with the Intel® Atom™ platform for a list of tutorials and their associated examples.
- Refer to the *High-Performance Embedded Programming with the Intel® Atom™ platform → Tutorial 2* for image convolution details as well as similar examples to the ones in this laboratory assignment.

Activities

First Activity: Image Convolution in C (50/50)

- Perform image convolution on a grayscale image for the two kernels shown in Fig. 1. This figure also depicts and input image as well as the resulting output images.
- You can use the code in *Tutorial 2* for image convolution (`imgconv.c, imgconv_fun.c, imgconv_fun.h, Makefile`) as a template for this Activity.
- To display the resulting images, you can use the `lab2a.m` script that runs in MATLAB® or GNU Octave (open-source version of MATLAB®). You can also use the script to generate input binary files from any picture you want to apply convolution to.

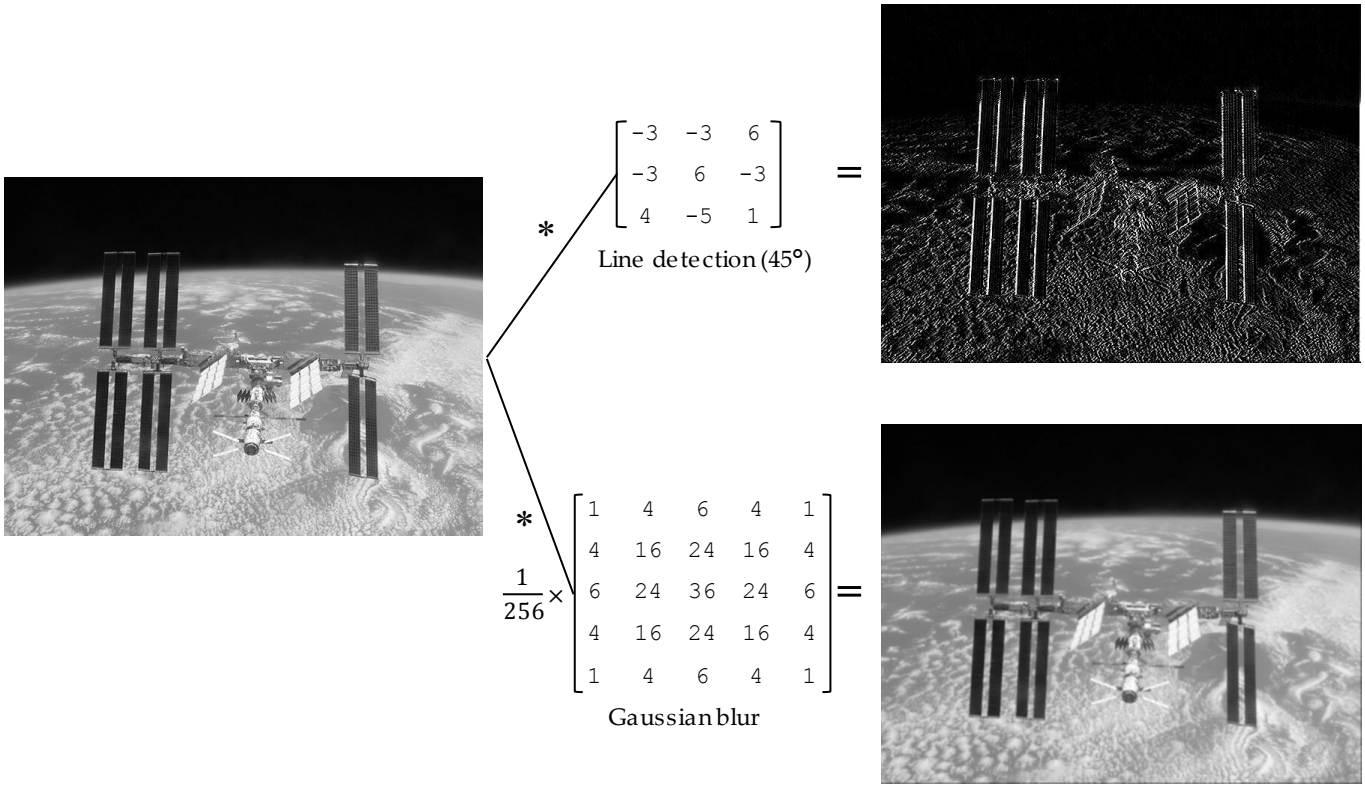

Figure 1. 2D convolution of a grayscale image for two different kernels: 3x3 line detection (45°) and 5x5 Gaussian blur. Note that when displaying, pixels values are restricted to [0, 255].

INSTRUCTIONS

- Write a `.c` program that reads in the parameter *t*, reads the binary input file (`.bif`), computes the respective 2D convolution, and stores the result in a binary output file (`.bof`).
  - t = 1: You use the 3x3 line detection (45°) kernel. The output `.bof` file should be named: `iss_a.bof`.
  - t = 2: You use the 5x5 Gaussian blur kernel. The output `.bof` file should be named: `iss_b.bof`.

- Considerations:
  - ✓ Input matrix: Read from an input binary file (`.bif`). You can use the provided `iss.bif` file that represents the 640x480 input image in Fig. 1. Each element is an unsigned 8-bit number.
  - ✓ Kernels: In general, they contain real-valued coefficients (like the Gaussian blur). Use `double` data type to represent them. You do not need to read the elements from a text file, you can just hard-code them in your main `.c` file.
    - Note: in the provided code in *Tutorial 2*, the kernel is stored in a linear array (raster scan version of the 2D array).
  - ✓ Output matrix: In general, it has real-valued elements. Use `double` data type to represent them.
    - The provided code in *Tutorial 2* needs to be modified in order to perform convolution with data of type `double`.
  - ✓ To store the output matrix in a `.bof` file, you need the output matrix elements to be `int` (32-bit signed integer).
    - As explained in *Tutorial 2*, the output matrix elements must be converted to `int` (32-bit signed integer). This is because the MATLAB® script will interpret the output matrix elements as 32-bit signed integers. So, you need to perform rounding of each matrix element followed by saturation.
      - Rounding: For each element `x`, you can use: `if (x >= 0) xr = x + 0.5; else xr = x-0.5;`
      - Saturation: You can use typecasting: `xi = (int) xr;` (using typecasting alone only does truncation + saturation).
  - ✓ Fig. 2 provides a graphical depiction of the different matrices involved in the process along with their data types.

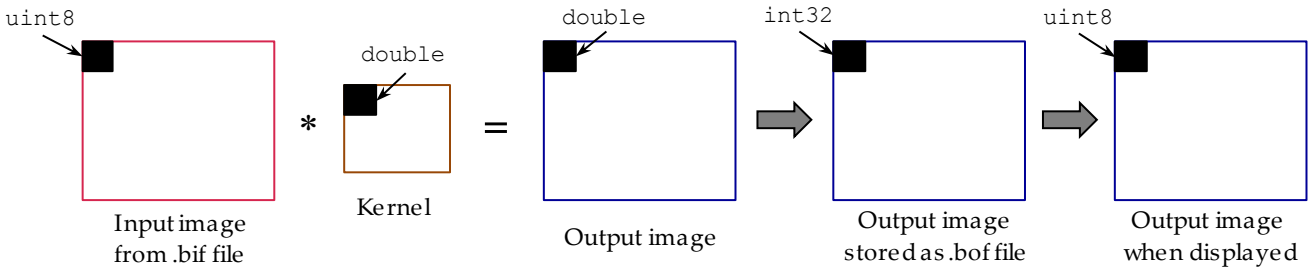

Figure 2. Image convolution process along with the data types involved at each step. This includes storing the output .bof file and displaying the resulting image.

- **Output matrix verification**: You need to verify the generated `.bof` files. You can do this via the `lab2a.m` script. Note that when displaying, we usually saturate the matrix to [0 255]. This is performed by the script.
  - ✓ Once you place the `.bof` files in the same folder as the script, run the script (select t = 1 or t = 2 in the script), and then Selector Operator '2' ("compare `.bof` file …"). The script will display the output image generated by MATLAB convolution as well as the output image generated by your `.c` code. They should match (a difference image is also displayed).

- Compile the code and execute the application on the DE2i-150 Board. Complete Table I.
  - ✓ Example: `./myconvimg 1`
  - ✓ Example: `./myconvimg 2`

- Take a screenshot of the software running in the Terminal (for 5x5 kernel0). It should show the computation time.
  - ✓ Your code should measure the computation time (only the actual computation portion) in us. This includes the rounding the output matrix elements (from `double` to `int`).

- Provided files: `lab2a.m, iss.jpg, iss.bif`.

TABLE I. COMPUTATION TIME (US) VS. PARAMETER *t*

Kernel	Computation Time (us)
3x3	
5x5	

SECOND ACTIVITY: NEURAL NETWORK LAYER IMPLEMENTATION IN C++ (50/50)

- In this activity, you will implement a neural network layer using a class (specifically a **functor**) in C++.

NEURAL NETWORK

- A 3-layer neural network (also called a Fully Connected Layer) is depicted in Fig. 3(a). The input layer represents the input values to the network. Fig. 3(b) depicts the inputs and output of the first neuron (index '1') in layer 3.
- Fig. 3(c) depicts an artificial neuron model. The neuron output (action potential a_j^l) results from applying an activation function to the membrane potential (z_j^l). The indices correspond to the first neuron (index '1') in layer l.
- The membrane potential z_j^l is a dot product between the inputs and the associated weights, to which a bias is then added.

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l, l > 1$$

- The action potential intensity of a neuron is denoted by a_j^l, and it is modeled as a scalar function (activation function) of z_j^l:

$$a_j^l = \sigma(z_j^l) = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right), l > 1$$

  - ✓ Common activation functions include:
    - Rectified Linear Unit (ReLU): $\sigma(z_j^l) = \max(0, z_j^l)$
    - Hyperbolic Tangent: $\sigma(z_j^l) = \tanh(z_j^l)$
    - Sigmoid function: $\sigma(z_j^l) = 1/\left(1 + e^{-z_j^l}\right)$

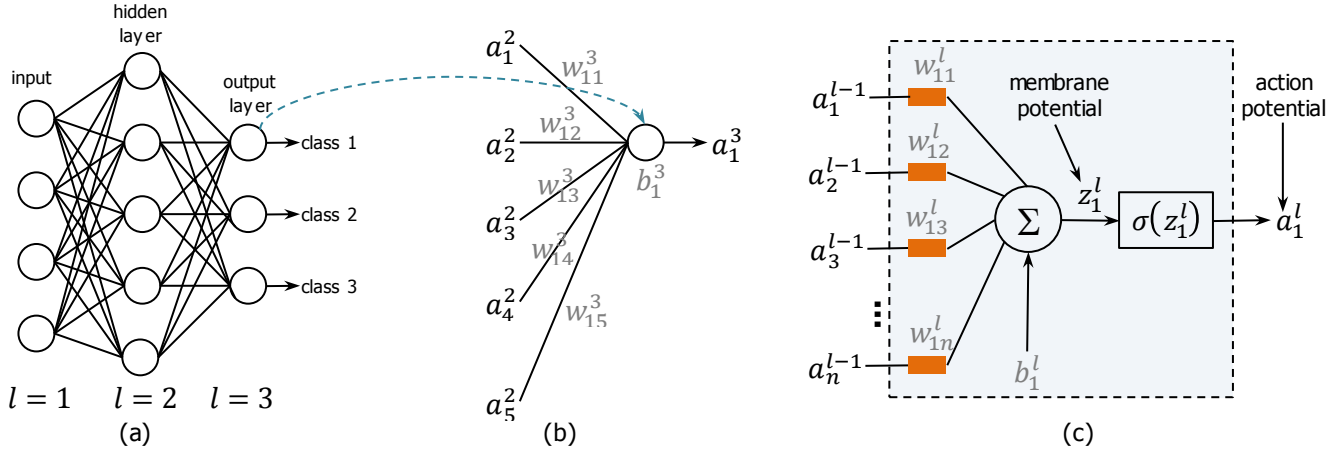

Figure 3. (a) 3-layer neural network. (b) First neuron (index '1') in layer *l*=3. (c) Artificial neuron model. The membrane potential is a sum of products (input activations by weights) to which a bias term is added. The neuron is the first neuron (index '1') in layer *l*. The input activations come from a previous layer (*l*-1).

- The output of a layer l can be described using a vectorized notation:

$$a^l = \sigma(z^l), \qquad z^l = w^l a^{l-1} + b^l, l > 1$$

Where:
w^l: weight matrix (NO rows by NI columns) of the layer l,
a^{l-1}: action potential vector (NI rows) of the previous layer l-1.
b^l: bias vector (NO rows) of the layer l.
z_l: membrane potential vector (NO rows) of the layer l.
a^l: action potential vector (NO rows) of the layer l.

- ✓ Fig. 4 depicts the matrix operation for z^l. NO: # of neurons in layer l. NI: # of inputs for each neuron in layer l.

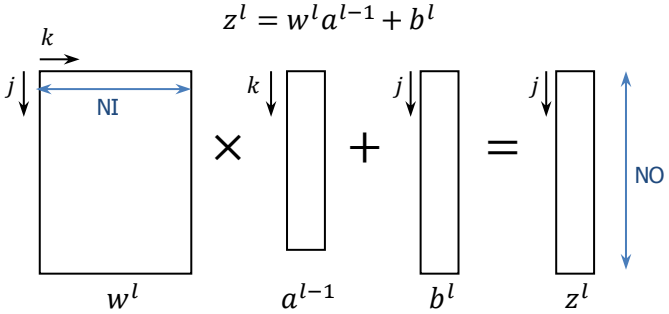

Figure 4. Matrix operation for the computation of all membrane potentials in layer *l*.

INSTRUCTIONS

- Write a C++ program (use `double` type for computations) that implements a neural network layer.
  - ✓ Neural network:
    - Parameters: `NI, NO`. You can hard-code them in `main()`.
    - Input data: w^l, a^{l-1}, b^l, σ (activation function). You may declare them in `main()` and then feed them to the neural network layer.
    - Output data: z^l and a^l. Your code should print these values.
  - ✓ You must use a **functor** to define the neural network layer (data members, constructors, and overloaded function calls). In `main()`, the following two lines **must** be present in your code:
    - `Network_Layex NL (NI,NO);` → Object definition and initialization.
      - The class here is called `Network_Layer`. When the object `NL` is defined, it should initialize `NI` and `NO`.
    - `NL(ap,w,b,af);` → treats the object as a function and executes the matrix operation.
      - Note that `ap` represents a^{l-1}, `w` is w^l, `b` is b^l, and `af` is the activation function σ (entered as an integer).
  - ✓ Tips:
    - Feel free to declare all member variables as public in the class.
    - Allocate memory properly for your data (note that `w` is a 2-D array)
  - ✓ Verification: Use the values specified in Fig. 5. Your result (z^l, a^l) should match those listed in Fig. 5.

$$\underbrace{\begin{bmatrix} 0.25 & 0.5 & -3.2 & -4.5 & -2.0 \\ 2.0 & 3.25 & 5.75 & 6.25 & 7.15 \\ 0.25 & -3.5 & 0.25 & 0.25 & 0.25 \\ 2.0 & 3.25 & 0.75 & -6.5 & 1.5 \end{bmatrix}}_{w^l} \times \underbrace{\begin{bmatrix} 2.5 \\ 3.0 \\ 2.5 \\ 1.5 \\ 1.0 \end{bmatrix}}_{a^{l-1}} + \underbrace{\begin{bmatrix} 2.0 \\ 1.5 \\ 2.5 \\ 3.5 \end{bmatrix}}_{b^l} = \underbrace{\begin{bmatrix} -12.625 \\ 47.150 \\ -6.125 \\ 11.875 \end{bmatrix}}_{z^l} \quad \underbrace{\sigma \begin{bmatrix} -12.625 \\ 47.150 \\ -6.125 \\ 11.875 \end{bmatrix}}_{z^l} = \underbrace{\begin{bmatrix} 0.000 \\ 47.150 \\ 0.000 \\ 11.875 \end{bmatrix}}_{a^l}$$

Figure 5. Testbed for your Neural Network Layer Implementation. NI=5, NO=4. Activation Function: ReLU.

- Compile the code and execute the application on the DE2i-150 Board.
  - ✓ Example: `./my_nnlayer`
- Take a screenshot of the software running in the Terminal. It should show the resulting values of z^l and a^l.

SUBMISSION

- Demonstration: In this Lab 2, the requested screenshot of the software routine running in the Terminal suffices.
  - ✓ If you prefer, you can request a virtual session (Webex) with the instructor and demo it (using a camera).
- Submit to Moodle (an assignment will be created):
  - ✓ Two .zip files (one for the 1st Activity and one for the 2nd Activity).
    - 1st Activity: The .zip file must contain the source files (`.c, .h, Makefile`), the output binary files (`.bof`) and the requested screenshot.
    - 2nd Activity: The .zip file must contain the source files (`.cpp, .h, Makefile`) and the requested screenshot.
  - ✓ The lab sheet (a PDF file) with the completed Table I.

TA signature: _____ **Date:** _____