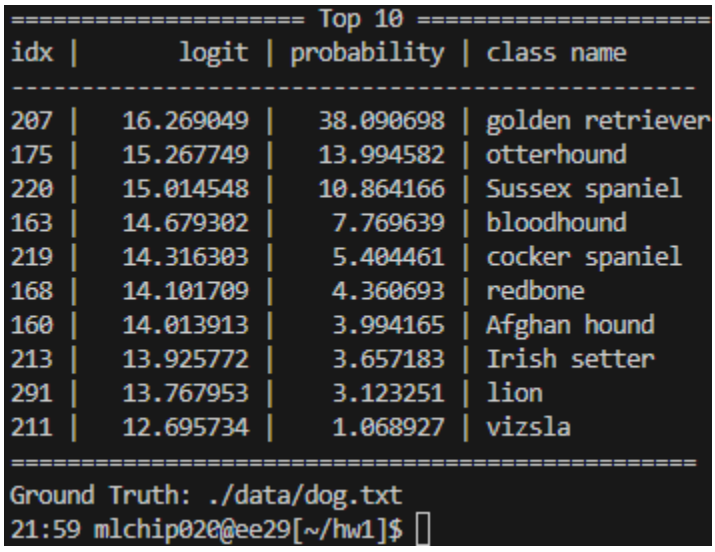
Machine Learning Intelligent Chip Design HW1

Name: 游鈞智　Student ID: 109511224

Part A: Simulation results demonstrate the predicted output for the provided input data.

- Output Result of ./data/dog.txt



```
==================== Top 10 ====================
idx |       logit | probability | class name
------------------------------------------------
207 |   16.269049 |   38.090698 | golden retriever
175 |   15.267749 |   13.994582 | otterhound
220 |   15.014548 |   10.864166 | Sussex spaniel
163 |   14.679302 |    7.769639 | bloodhound
219 |   14.316303 |    5.404461 | cocker spaniel
168 |   14.101709 |    4.360693 | redbone
160 |   14.013913 |    3.994165 | Afghan hound
213 |   13.925772 |    3.657183 | Irish setter
291 |   13.767953 |    3.123251 | lion
211 |   12.695734 |    1.068927 | vizsla
================================================
Ground Truth: ./data/dog.txt
21:59 mlchip020@ee29[~/hw1]$
```

- Output Result of ./data/cat.txt

```
==================== Top 10 ====================
idx |       logit | probability | class name
------------------------------------------------
285 |   19.915667 |   96.042755 | Egyptian cat
281 |   15.842955 |    1.635718 | tabby
282 |   15.629088 |    1.320771 | tiger cat
287 |   14.637820 |    0.490146 | lynx
728 |   14.275681 |    0.341232 | plastic bag
330 |   12.477574 |    0.056512 | wood rabbit
331 |   12.003444 |    0.035175 | hare
457 |   10.888266 |    0.011532 | bow tie
463 |   10.566185 |    0.008357 | bucket
335 |   10.416719 |    0.007197 | fox squirrel
================================================
Ground Truth: ./data/cat.txt
22:01 mlchip020@ee29[~/hw1]$
```

Part B: Implementation Approach

1. In this assignment, apart from SC_AlexNet, other network components (including convolution layers, fully connected layers, etc.) are implemented using C++ classes. Each class of network components contains a forward function used to implement the forward propagation of AlexNet. If the class includes trainable parameters, there will be an additional function called read_weights() to read pretrained weights, as well as show_weights() to print out the loaded parameters.
   Below is a screenshot of the Convolution class function members.

```
public:
    Convolution(string module_name, int input_channels, int output_channels, int kernel_size, int stride, int padding);
    void read_weights();
    void show_weights();
    vector<vector<vector<float>>> forward(vector<vector<vector<float>>> input_image);
```

2. To confirm the parameters of each layer (padding stride, etc.), I referred to the model information provided by the TAs as well as the source code of PyTorch AlexNet. I inferred the detailed parameters of each layer.
   Below are the definitions of each layer as inferred by me:
   Convolution(in_channels, out_channels, kernel_size, stride, padding)
   MaxPooling(kernel_size, stride)
   AdaptiveAvgPooling(output_height, output_width)
   FullyConnected(input_size, output_size)

```
conv1("conv1", 3, 64, 11, 4, 2),
relu1("relu1"),
pool1("pool1", 3, 2),
conv2("conv2", 64, 192, 5, 1, 2),
relu2("relu2"),
pool2("pool2", 3, 2),
conv3("conv3", 192, 384, 3, 1, 1),
relu3("relu3"),
conv4("conv4", 384, 256, 3, 1, 1),
relu4("relu4"),
conv5("conv5", 256, 256, 3, 1, 1),
relu5("relu5"),
pool5("pool5", 3, 2),
adaptive_avg_pool("adaptive_avg_pool", 6, 6),
flatten("flatten"),
fc6("fc6", 9216, 4096),
relu6("relu6"),
fc7("fc7", 4096, 4096),
relu7("relu7"),
fc8("fc8", 4096, 1000)
```

3. In the end, I packaged all the layer components into an sc_module. By inputting an image stored in a three-dimensional sc_vector, the output consists of two sc_vectors, each with a dimension of 1000. One represents the value (logit), and the other represents the output probability after softmax processing.

```
SC_MODULE(SC_AlexNet)
{
    sc_vector<sc_vector<sc_vector<sc_in<float>>>> input_image;
    sc_vector<sc_out<float>> output_prob;
    sc_vector<sc_out<float>> output_logit;
```

Part C: Challenges Faced

1. For this assignment, it was evident that a three-dimensional sc_vector was a more suitable data type. After quite a bit of research, I discovered that it could also be wrapped into a three-dimensional sc_vector using a method similar to C++ vectors. This approach avoided the tedious process of sending a 1D sc_vector each time and then resizing it into 3D, only to flatten it back into 1D during output.

```
sc_vector<sc_vector<sc_vector<sc_in<float>>>> input_image;
```

2. Since it is necessary to add all sc_in to the sensitive list, and all input sc_signals must be connected to sc_in, sc_out one by one, it is essential to define the dimensions of the input. The first method that came to mind for me is as follows:

```
sc_vector<sc_vector<sc_vector<sc_in<float>>>> input{"input", 3, [](char const *name, size_t idx) -> sc_vector<sc_vector<sc_in<float>>> *
                                                    {
                                                        return new sc_vector<sc_vector<sc_in<float>>>(name, 224, [](char const *name, size_t idx) -> sc_vector<sc_in<float>> *
                                                                                                      { return new sc_vector<sc_in<float>>(name, 224); });
                                                    }};
```

The lambda function provided initializes each element of the sc_vector with another sc_vector of sc_in<float> elements, creating a nested structure to accommodate the multidimensional nature of the input data.

Due to its complexity in writing and for the convenience of debugging later, it was ultimately decided to adjust the dimensions of sc_in and sc_out within the SC_CTOR using a for loop combined with init() function.

3. Since Convolution requires padding and may result in filter coverage extending beyond the feature map's boundaries, initially, I intended to calculate how much the filter would exceed and consider padding the input feature map with zeros around it. However, this would undoubtedly increase the complexity of the design.

```
int input_height_index = j * stride + m - padding;
int input_width_index = k * stride + n - padding;
if (input_height_index >= 0 && input_height_index < input_height && input_width_index >= 0 && input_width_index < input_width)
    sum += input_image[l][input_height_index][input_width_index] * weights[i][l][m][n];
```

   Considering that the padding mode is zero padding and that zeros would be filled in if the filter coverage extends beyond, I decided to calculate it accordingly. During computation, if it exceeds the calculation range (e.g., filter coverage), it will be skipped directly. By adopting this approach, both of these issues can be addressed simultaneously.

Part D: Insights

1. The first observation I made was that there were slight discrepancies between my model's output values and the sample outputs provided by the TA, as well as the results from PyTorch execution. Although these variances did not affect the determination of the Top K accuracy, they are still worth considering.

   Since both weights and biases undergo rounding, it is normal to have slight differences compared to PyTorch. I suspect that the variance in numerical values between my results and those of the TA could be due to float or double precision. In future assignments, it may be worth considering using sc_fixed to define floating-point numbers for calculations.

2. The second observation is regarding the characteristics of SystemC. In SystemC, code executes by simultaneously manipulating values and generating results in parallel, similar to other Hardware Description Languages (HDLs) like Verilog. For instance, assigning a value to an sc_signal at the same moment and then immediately using read() will not provide the written value; it requires waiting until the next moment to do so.

Part E: Usage

After compiling, an executable named "run" will be generated. When executing, you need to append the path of the image at the end.

For example: ./run "./data/dog.txt"

Since only two datasets are provided this time, I have directly added options for "cat" and "dog" in the MakeFile for the convenience of TAs.

```
all:
    clear
    g++ -I . -I  $(INC_DIR) -L . -L $(LIB_DIR) -o $(O) $(C) $(LIB) $(RPATH)
    echo "Usage: ./run <input_file>"

cat:
    clear
    g++ -I . -I  $(INC_DIR) -L . -L $(LIB_DIR) -o $(O) $(C) $(LIB) $(RPATH)
    ./run "./data/cat.txt"

dog:
    clear
    g++ -I . -I  $(INC_DIR) -L . -L $(LIB_DIR) -o $(O) $(C) $(LIB) $(RPATH)
    ./run "./data/dog.txt"
```