report.md

Lab09

2014-17831 김재원

`benchmark.sh`

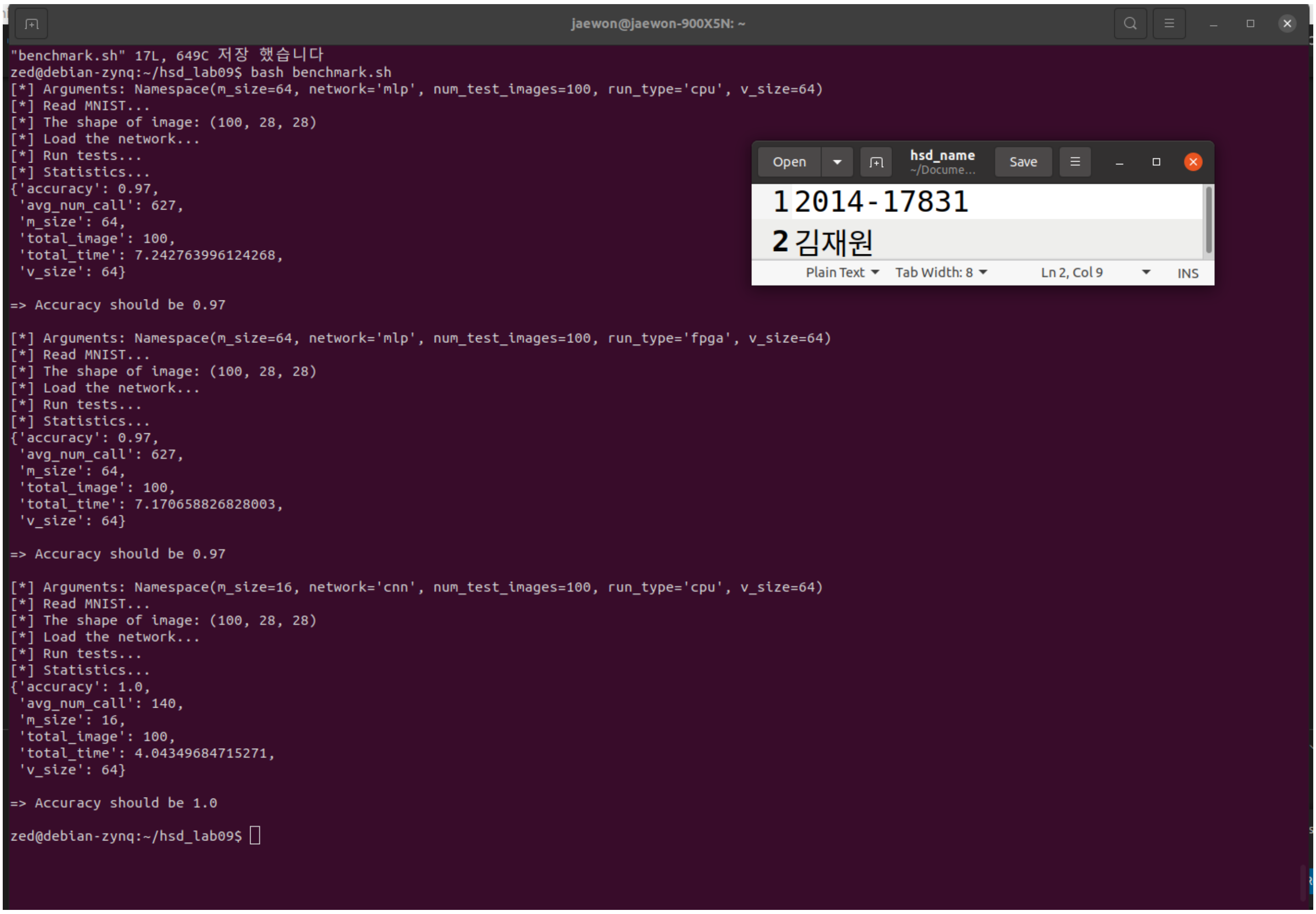
Input)

```
python eval.py --num_test_images 100 --m_size 64 --v_size 64 --network mlp --run_type cpu
echo -e '\n=> Accuracy should be 0.97\n'

# MV - Acc: 0.97
python eval.py --num_test_images 100 --m_size 64 --v_size 64 --network mlp --run_type fpga
echo -e '\n=> Accuracy should be 0.97\n'

# Conv Lowering(CPU) - Acc: 1.0
python eval.py --num_test_images 100 --v_size 64 --network cnn --run_type cpu
echo -e '\n=> Accuracy should be 1.0\n'
```

Results on FPGA)



Results on Linux)

```
root@563a5aed49b2:~/hsd_lab09# bash benchmark.sh
[*] Arguments: Namespace(m_size=64, network='mlp', num_test_images=100, run_type='cpu', v_size=64)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
```

```
[*] Statistics...
{'accuracy': 0.97,
 'avg_num_call': 627,
 'm_size': 64,
 'total_image': 100,
 'total_time': 0.4345870018005371,
 'v_size': 64}

=> Accuracy should be 0.97

[*] Arguments: Namespace(m_size=64, network='mlp', num_test_images=100, run_type='fpga', v_size=64)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.97,
 'avg_num_call': 627,
 'm_size': 64,
 'total_image': 100,
 'total_time': 0.43550682067871094,
 'v_size': 64}

=> Accuracy should be 0.97

[*] Arguments: Namespace(m_size=16, network='cnn', num_test_images=100, run_type='cpu', v_size=64)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 140,
 'm_size': 16,
 'total_image': 100,
 'total_time': 0.4194941520690918,
 'v_size': 64}

=> Accuracy should be 1.0
```

Evaluations

`num_test_images = 1, v_size = 8`

```
python eval.py --num_test_images 1 --v_size 8 --network cnn --run_type cpu
```

```
[*] Arguments: Namespace(m_size=16, network='cnn', num_test_images=1, run_type='cpu', v_size=8)
[*] Read MNIST...
[*] The shape of image: (1, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 1188,
 'm_size': 16,
 'total_image': 1,
 'total_time': 0.007005929946899414,
 'v_size': 8}
```

`num_test_images = 1, v_size = 64`

```
python eval.py --num_test_images 1 --v_size 64 --network cnn --run_type cpu
```

```
[*] Arguments: Namespace(m_size=16, network='cnn', num_test_images=1, run_type='cpu', v_size=64)
[*] Read MNIST...
[*] The shape of image: (1, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
```

```
 'avg_num_call': 140,
 'm_size': 16,
 'total_image': 1,
 'total_time': 0.04099106788635254,
 'v_size': 64}
```

num_test_images = 10, v_size = 8

```
python eval.py --num_test_images 10 --v_size 8 --network cnn --run_type cpu
```

```
[*] Arguments: Namespace(m_size=16, network='cnn', num_test_images=10, run_type='cpu', v_size=64)
[*] Read MNIST...
[*] The shape of image: (10, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 140,
 'm_size': 16,
 'total_image': 10,
 'total_time': 0.41094207763671875,
 'v_size': 64}
```

num_test_images = 100, v_size = 64

```
python eval.py --num_test_images 100 --v_size 64 --network cnn --run_type cpu
```

```
[*] Arguments: Namespace(m_size=16, network='cnn', num_test_images=100, run_type='cpu', v_size=64)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 140,
 'm_size': 16,
 'total_image': 100,
 'total_time': 4.046089172363281,
 'v_size': 64}
```

num_test_images = 1000, v_size = 64

```
python eval.py --num_test_images 1000 --v_size 64 --network cnn --run_type cpu
```

```
[*] Arguments: Namespace(m_size=16, network='cnn', num_test_images=1000, run_type='cpu', v_size=64)
[*] Read MNIST...
[*] The shape of image: (1000, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.98,
 'avg_num_call': 140,
 'm_size': 16,
 'total_image': 1000,
 'total_time': 40.87884497642517,
 'v_size': 64}
```

num_test_images = 10000, v_size = 64

```
python eval.py --num_test_images 10000 --v_size 64 --network cnn --run_type cpu
```

```
[*] Arguments: Namespace(m_size=16, network='cnn', num_test_images=10000, run_type='cpu', v_size=64)
[*] Read MNIST...
[*] The shape of image: (10000, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
```

```
{'accuracy': 0.98,
 'avg_num_call': 140,
 'm_size': 16,
 'total_image': 10000,
 'total_time': 413.2099578380585,
 'v_size': 64}
```

`num_test_images = 10000, v_size = 32`

```
python eval.py --num_test_images 10000 --v_size 32 --network cnn --run_type cpu
```

```
[*] Arguments: Namespace(m_size=16, network='cnn', num_test_images=10000, run_type='cpu', v_size=32)
[*] Read MNIST...
[*] The shape of image: (10000, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.98,
 'avg_num_call': 277,
 'm_size': 16,
 'total_image': 10000,
 'total_time': 136.75659918785095,
 'v_size': 32}
```

`num_test_images = 10000, v_size = 16`

```
python eval.py --num_test_images 10000 --v_size 16 --network cnn --run_type cpu
```

```
[*] Arguments: Namespace(m_size=16, network='cnn', num_test_images=10000, run_type='cpu', v_size=16)
[*] Read MNIST...
[*] The shape of image: (10000, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.98,
 'avg_num_call': 553,
 'm_size': 16,
 'total_image': 10000,
 'total_time': 69.15588092803955,
 'v_size': 16}
```

`num_test_images = 10000, v_size = 8`

```
python eval.py --num_test_images 10000 --v_size 8 --network cnn --run_type cpu
```

```
[*] Arguments: Namespace(m_size=16, network='cnn', num_test_images=10000, run_type='cpu', v_size=8)
[*] Read MNIST...
[*] The shape of image: (10000, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.98,
 'avg_num_call': 1188,
 'm_size': 16,
 'total_image': 10000,
 'total_time': 68.88850593566895,
 'v_size': 8}
```

Implementation

`largeMM()`

- 이 함수의 구현은 2주차에 구현했던 `largeMV()` 에서 matrix를 assign하는 부분을 가져와 가로와 세로에 해당되는 부분만 변경했다.
- 매우 헷갈렸지만, 이 질의응답의 댓글을 참고하여 가로와 세로를 변경하니 쉽게 구현할 수 있었다.

```
// ...

/*
  * M1: num_output * num_input
  * M2: num_input * num_matrix2
  * out: num_output * num_matrix2
*/

// 1) Assign a m1
for(int row = 0; row < v_size_; ++row)
{
  memset(m1 + (v_size_*row), 0, sizeof(float) * v_size_);
  if(row < block_row)
  {
    memcpy(m1 + (v_size_*row), weight_mat + (num_input*(i+row)) + j, sizeof(float) * block_col_1);
  }
}

// 2) Assign a m2
for(int row = 0; row < v_size_; ++row)
{
  memset(m2 + (v_size_*row), 0, sizeof(float) * v_size_);
  if(row < block_col_1)
  {
    memcpy(m2 + (v_size_*row), input_mat + (num_matrix2*(j+row)) + k, sizeof(float) * block_col_2);
  }
}
// ...
```

`convLowering()`

- Lab 09 자료의 10번 슬라이드를 많이 참고하여 구현하였다.
- 처음에는 무엇을 하고자 하는지조차 헷갈려서 어려움을 겪었지만, 10번 슬라이드에서 D4, D5, D7, D8에 해당하는 input이 들어오는 경우를 따져보다 보니 input의 어떤 index에 해당하는 값이 new_input의 어떤 index로 이동하게 되는지 파악할 수 있게 되어 구현할 수 있었다.
- 특히, conv_channel, input_channel, conv_height, conv_width 등의 변수들이 미리 (순서대로) 정의되어 있어 각각이 무엇을 의미하고, 어디에 쓰이는지 파악하는 데에 많은 도움이 되었다.
- 해당 정보를 이용하여 차례대로 for loop를 구성하니 오류 없이 구현할 수 있었다.
- new input을 구현하면서 헷갈렸던 부분은 convolution 크기 (height/width)를 고려해서 outer for loop 두 개의 boundary condition을 정해줘야 할 뿐 아니라, new_input에 값을 저장할 때도 이 부분을 고려해서 new_input의 최종 width를 구해야 했다는 점이다.
  - 후자의 경우 new_inputs[][이 부분]에 해당하는 값을 구할 때 new_inputs matrix의 width만큼 height를 곱해주어야 되기 때문이다.

```
// ...
// new_weights
for(int c = 0; c < conv_channel; c++) {
  for(int i = 0; i < input_channel; i++) {
    for(int h = 0; h < conv_height; h++) {
      for(int w = 0; w < conv_width; w++) {
        new_weights[c][(i*conv_height*conv_width) + (h*conv_width+w)] = cnn_weights[c][i][h][w];
      }
    }
  }
}

// new inputs
for(int h = 0; h <= (input_height-conv_height); h++) {
  for(int w = 0; w <= (input_width-conv_width); w++) {
    for(int i = 0; i < input_channel; i++) {
      for(int y = 0; y < conv_height; y++) {
        for(int x = 0; x < conv_width; x++) {
          new_inputs[(i*conv_height*conv_width) + (y*conv_width+x)][h*(input_width-conv_width+1)+w] = inputs[i][h+y][w
        }
      }
    }
  }
}
```