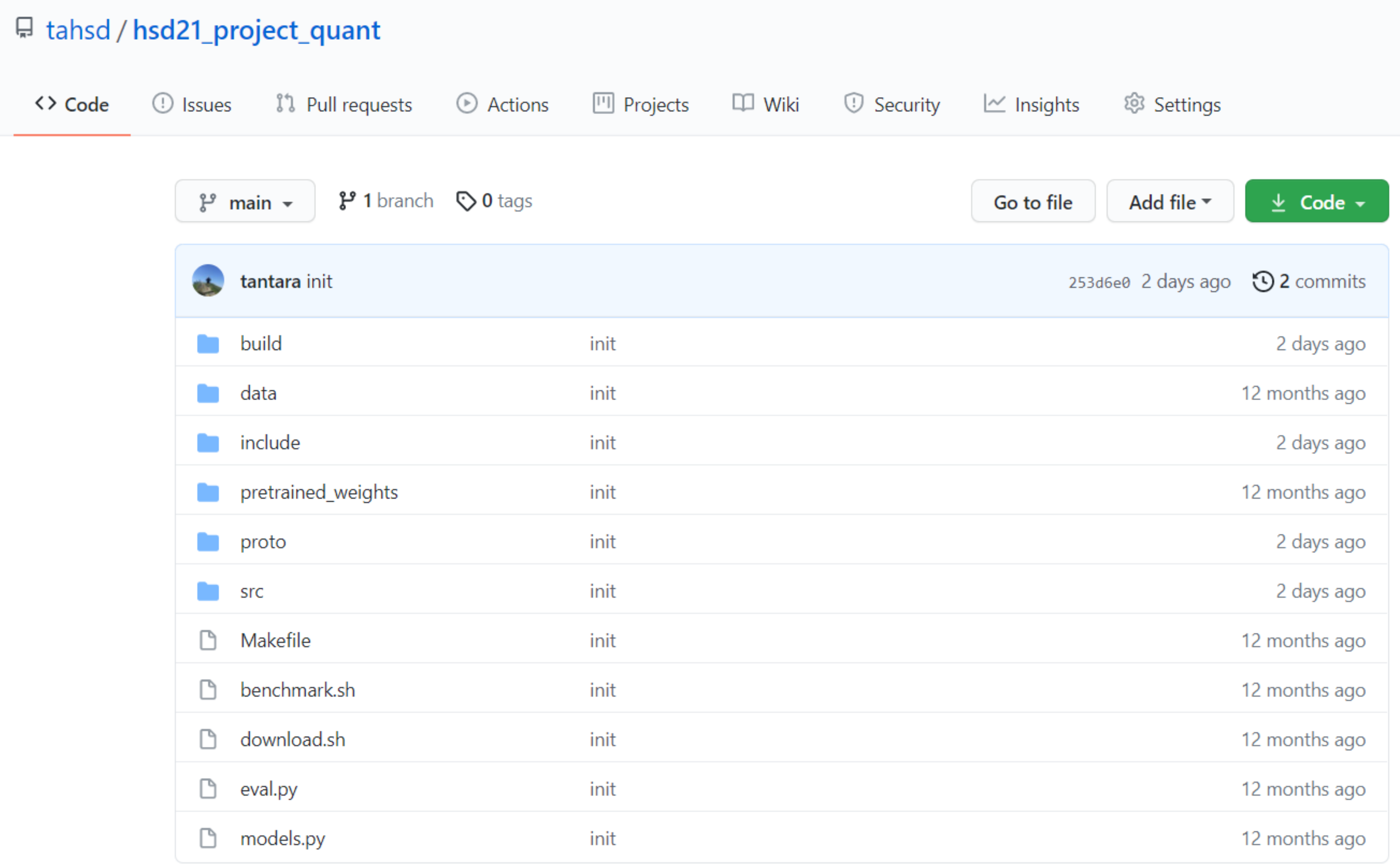
Optimization – Quantization (optional)

- Quantization on Board

Computing Memory Architecture Lab.

Quantization on Board

New github repo

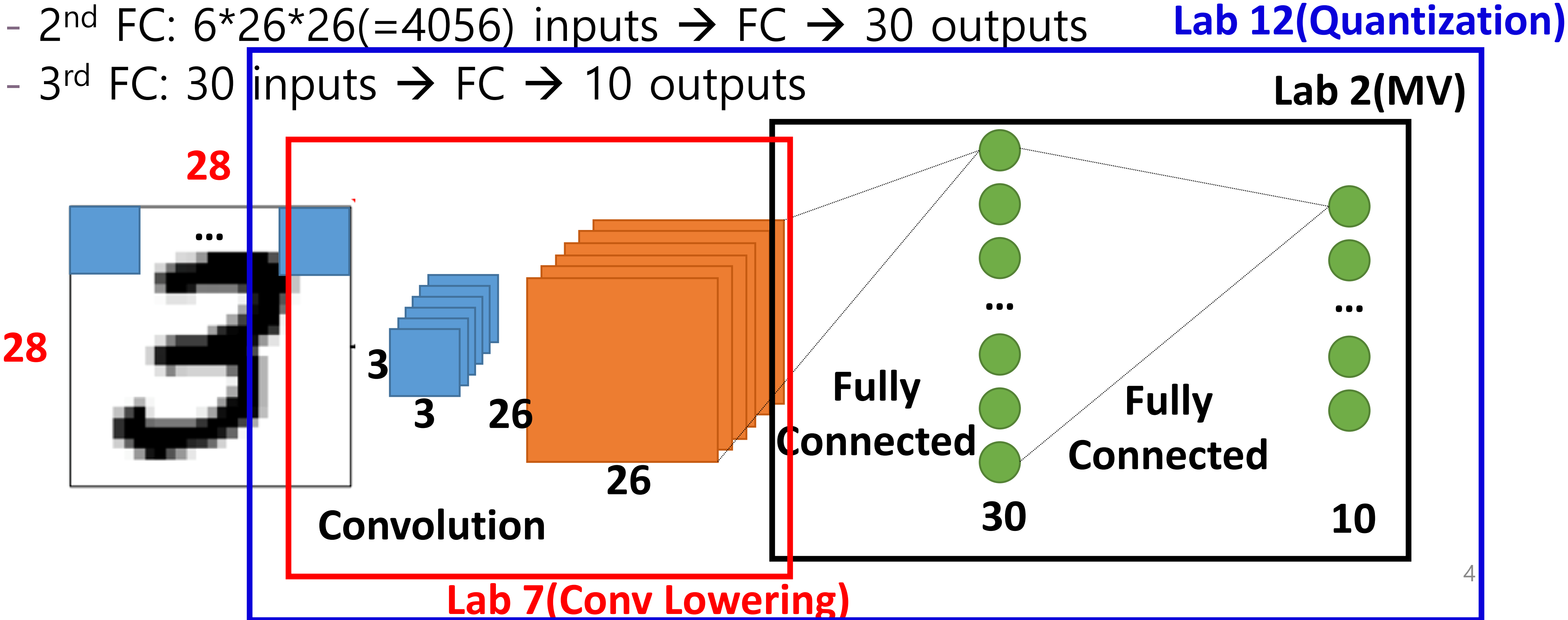


tantara init 253d6e0 2 days ago 2 commits

build	init	2 days ago
data	init	12 months ago
include	init	2 days ago
pretrained_weights	init	12 months ago
proto	init	2 days ago
src	init	2 days ago
Makefile	init	12 months ago
benchmark.sh	init	12 months ago
download.sh	init	12 months ago
eval.py	init	12 months ago
models.py	init	12 months ago

(pretrained) Convolutional Network Network(CNN)

- Input: **28x28** pixels → **6 3x3** Conv → **30** values → **10** values
  - 1st Conv: **28x28** inputs → 6 3x3 Conv → 6 26x26 outputs
  - 2nd FC: 6*26*26(=4056) inputs → FC → 30 outputs
  - 3rd FC: 30 inputs → FC → 10 outputs



[Jacob, 2018]

Review: Google's Int8 Solution

- Goal: running int8 neural networks on CPUs supporting int8 SIMD instructions
- Conversion, i.e., quantization from floating to 8-bit unsigned integer

Real value r

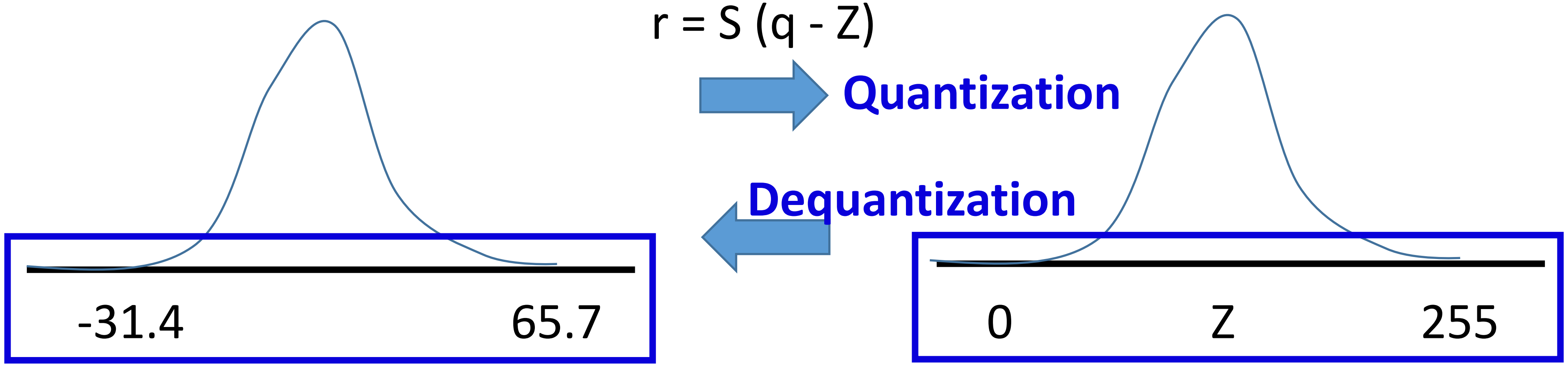
32-bit floating

Quantized value q

8-bit unsigned integer with 256 levels (0~255)

$q = \mathrm{ceil}(r / S) + Z$

$r = S (q - Z)$



[Jacob, 2018]

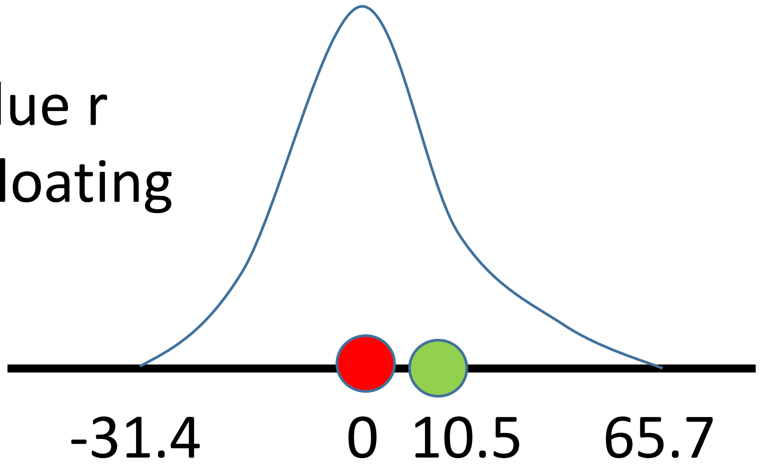
Review: Quantization Example

- Calculating scale (S) and zero (Z) for int8

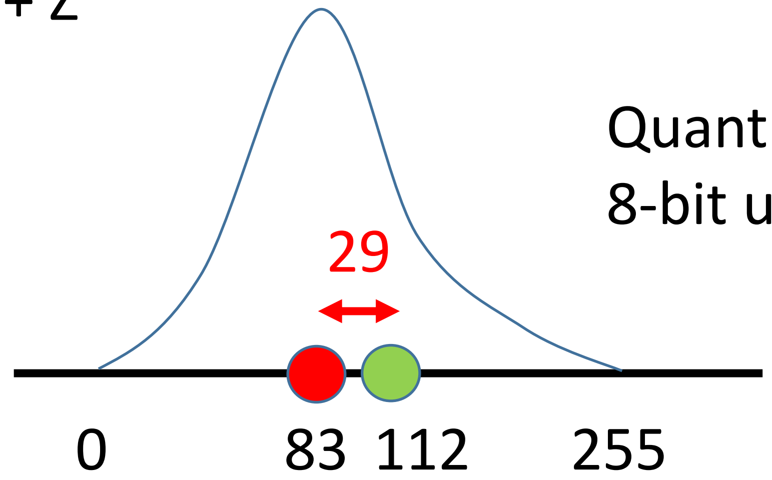
-31.4 → 0	-31.4 = S*(0 – Z)	SZ=31.4	Z=ceil(31.4/0.38)=83
65.7 → 255	65.7 = S*(255 –Z)	65.7+31.4=255S	S=97.1/255=0.38

- Example: 10.5 → ceil(10.5/0.38)+83=29+83=112

Real value r
32-bit floating



q = ceil(r / S) + Z
r = S (q - Z)



Quantized value q
8-bit unsigned int

[Jacob, 2018]

Review: Integer Only Computation

- Matrix multiplication, $r_3 = r_1 * r_2$
  - Weight matrix r_1 and activation matrix r_2

Zero offset

$$\boxed{r_\alpha^{(i,j)} = S_\alpha (q_\alpha^{(i,j)} - Z_\alpha)}$$

$$S_3(q_3^{(i,k)} - Z_3) = \sum_{j=1}^{N} S_1(q_1^{(i,j)} - Z_1) S_2(q_2^{(j,k)} - Z_2)$$

$$q_3^{(i,k)} = Z_3 + M \sum_{j=1}^{N} (q_1^{(i,j)} - Z_1)(q_2^{(j,k)} - Z_2)$$

$$\boxed{M := \frac{S_1 S_2}{S_3}}$$

Scaling(=dequantization)

What if Z_1 and Z_2 are zero?

$$q_3^{(i,k)} = Z_3 + M \left(N Z_1 Z_2 - Z_1 a_2^{(k)} - Z_2 \bar{a}_1^{(i)} + \sum_{j=1}^{N} q_1^{(i,j)} q_2^{(j,k)} \right)$$

$$a_2^{(k)} := \sum_{j=1}^{N} q_2^{(j,k)}, \quad \bar{a}_1^{(i)} := \sum_{j=1}^{N} q_1^{(i,j)}$$

$$\sum_{j=1}^{N} q_1^{(i,j)} q_2^{(j,k)}$$

Solution: 7b quantization + zero offset

- Calculating scale (S) and zero (Z) for 7b

-31.4 → 0
65.7 → 127

$-31.4 = S*(0 - Z)$
$65.7 = S*(127 - Z)$

$SZ=31.4$
$65.7+31.4=127S$

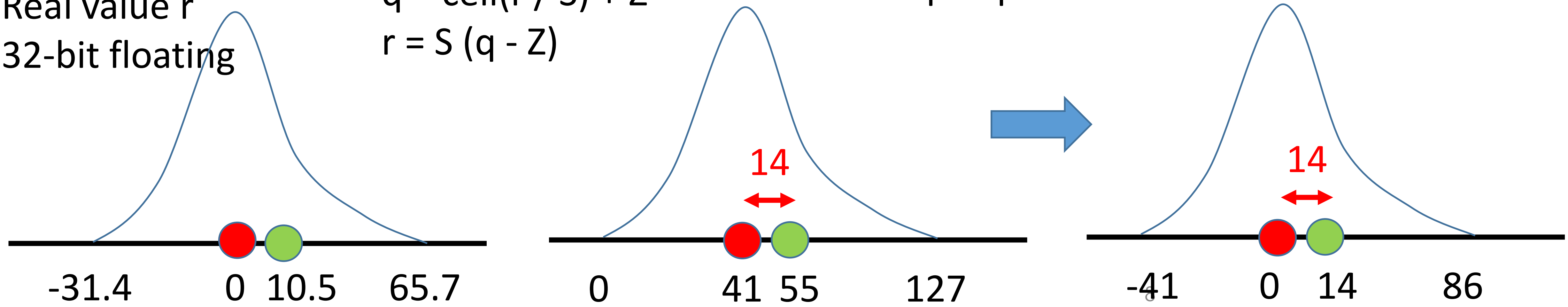
$Z=ceil(31.4/0.76)=41$
$S=97.1/127=0.76$

- Example: 10.5 → ceil(10.5/0.76)+41=14+41=55

Real value r
32-bit floating

$q = ceil(r / S) + Z$
$r = S (q - Z)$

$q' = q-Z$



Install

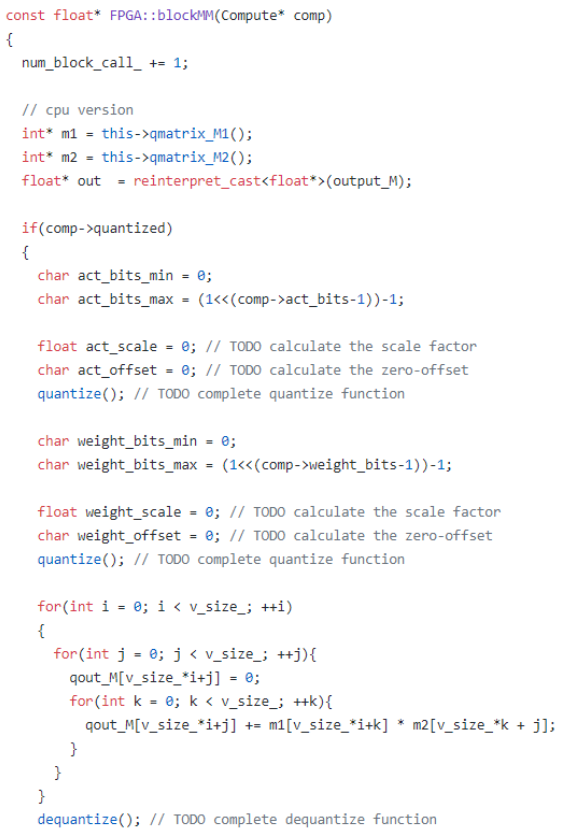
- **Connect to the internet with LAN cable**
- Dependency download
  ```
  $ sudo apt-get update -y
  $ sudo apt-get install -y libprotobuf-dev protobuf-compiler python python-numpy
  ```
- Code download
  ```
  $ git clone https://github.com/tahsd/hsd21_project_quant.git
  ```
- Dataset download
  ```
  $ bash download.sh
  ```

Run

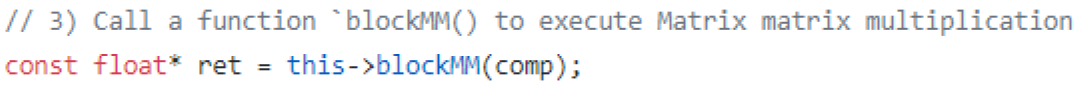
- Edit `src/fpga_api_on_cpu.cpp`
  - void quantize(…), dequantize(…)
  - Also edit `src/fpga_api.cpp` for optimization
- Build
  - $ make
- Evaluate
  - $ sudo python eval.py --num_test_images 100 --m_size 8 --v_size 8 --network cnn --run_type cpu **--quantized**
- Options
  - num_test_images : 1~10000
  - m_size, v_size : 8
  - network: cnn
  - run_type: cpu(tutorial), fpga(optimized version)
    - fpga uses the IP that you implemented
  - **quantized: enable(8-bit) or disable(32-bit, full precision)**

Implement Quantization - CPU

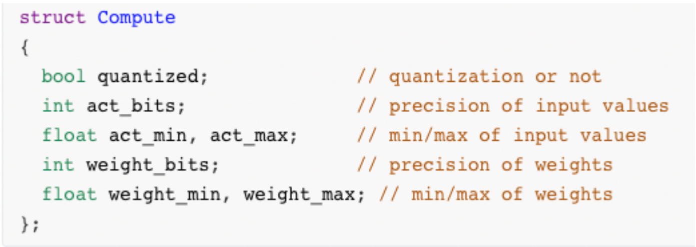
- Edit `src/fpga_api_on_cpu.cpp`



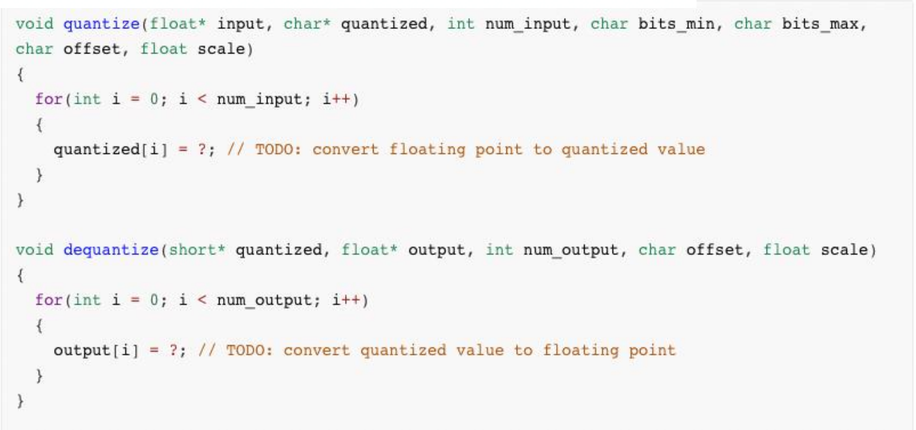
```cpp
const float* FPGA::blockMM(Compute* comp)
{
  num_block_call_ += 1;

  // cpu version
  int* m1 = this->qmatrix_M1();
  int* m2 = this->qmatrix_M2();
  float* out  = reinterpret_cast<float*>(output_M);

  if(comp->quantized)
  {
    char act_bits_min = 0;
    char act_bits_max = (1<<(comp->act_bits-1))-1;

    float act_scale = 0; // TODO calculate the scale factor
    char act_offset = 0; // TODO calculate the zero-offset
    quantize(); // TODO complete quantize function

    char weight_bits_min = 0;
    char weight_bits_max = (1<<(comp->weight_bits-1))-1;

    float weight_scale = 0; // TODO calculate the scale factor
    char weight_offset = 0; // TODO calculate the zero-offset
    quantize(); // TODO complete quantize function

    for(int i = 0; i < v_size_; ++i)
    {
      for(int j = 0; j < v_size_; ++j){
        qout_M[v_size_*i+j] = 0;
        for(int k = 0; k < v_size_; ++k){
          qout_M[v_size_*i+j] += m1[v_size_*i+k] * m2[v_size_*k + j];
        }
      }
    }
    dequantize(); // TODO complete dequantize function
```



```cpp
// 3) Call a function `blockMM() to execute Matrix matrix multiplication
const float* ret = this->blockMM(comp);
```



```cpp
struct Compute
{
  bool quantized;              // quantization or not
  int act_bits;                // precision of input values
  float act_min, act_max;      // min/max of input values
  int weight_bits;             // precision of weights
  float weight_min, weight_max; // min/max of weights
};
```

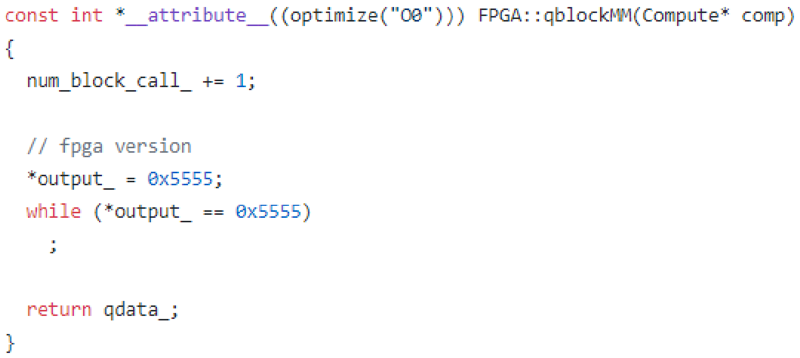


```cpp
void quantize(float* input, char* quantized, int num_input, char bits_min, char bits_max,
char offset, float scale)
{
  for(int i = 0; i < num_input; i++)
  {
    quantized[i] = ?; // TODO: convert floating point to quantized value
  }
}

void dequantize(short* quantized, float* output, int num_output, char offset, float scale)
{
  for(int i = 0; i < num_output; i++)
  {
    output[i] = ?; // TODO: convert quantized value to floating point
  }
}
```

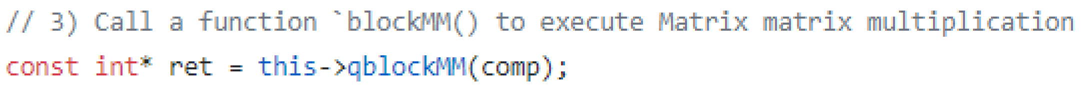
- [Note] Fill other functions `convLowering`, etc…

Implement Quantization - FPGA

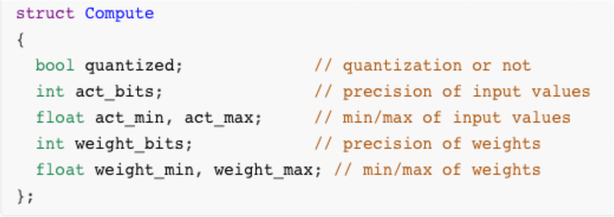
- Edit `src/fpga_api.cpp`



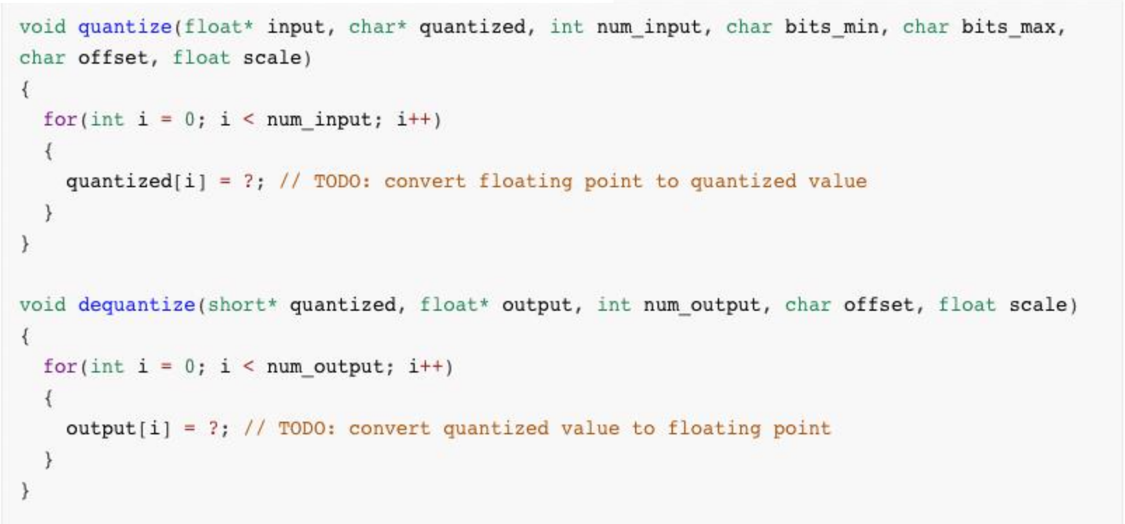
```
const int *__attribute__((optimize("O0"))) FPGA::qblockMM(Compute* comp)
{
  num_block_call_ += 1;

  // fpga version
  *output_ = 0x5555;
  while (*output_ == 0x5555)
    ;

  return qdata_;
}
```



```
// 3) Call a function `blockMM() to execute Matrix matrix multiplication
const int* ret = this->qblockMM(comp);
```



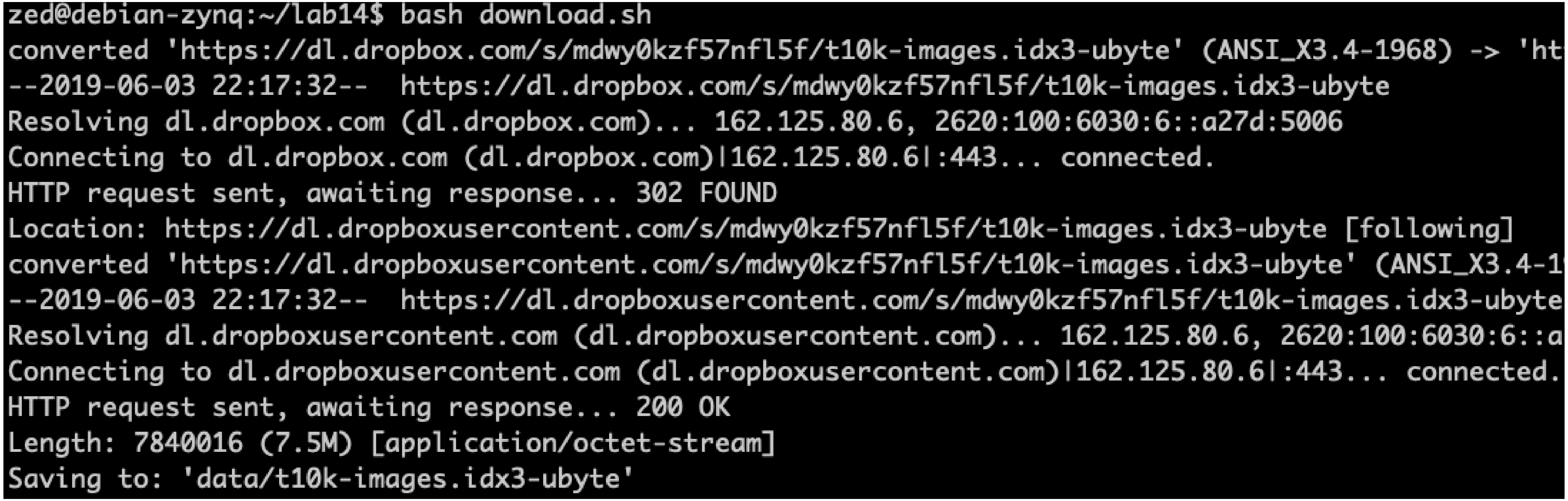
```
struct Compute
{
  bool quantized;             // quantization or not
  int act_bits;               // precision of input values
  float act_min, act_max;     // min/max of input values
  int weight_bits;            // precision of weights
  float weight_min, weight_max; // min/max of weights
};
```



```
void quantize(float* input, char* quantized, int num_input, char bits_min, char bits_max,
char offset, float scale)
{
  for(int i = 0; i < num_input; i++)
  {
    quantized[i] = ?; // TODO: convert floating point to quantized value
  }
}

void dequantize(short* quantized, float* output, int num_output, char offset, float scale)
{
  for(int i = 0; i < num_output; i++)
  {
    output[i] = ?; // TODO: convert quantized value to floating point
  }
}
```

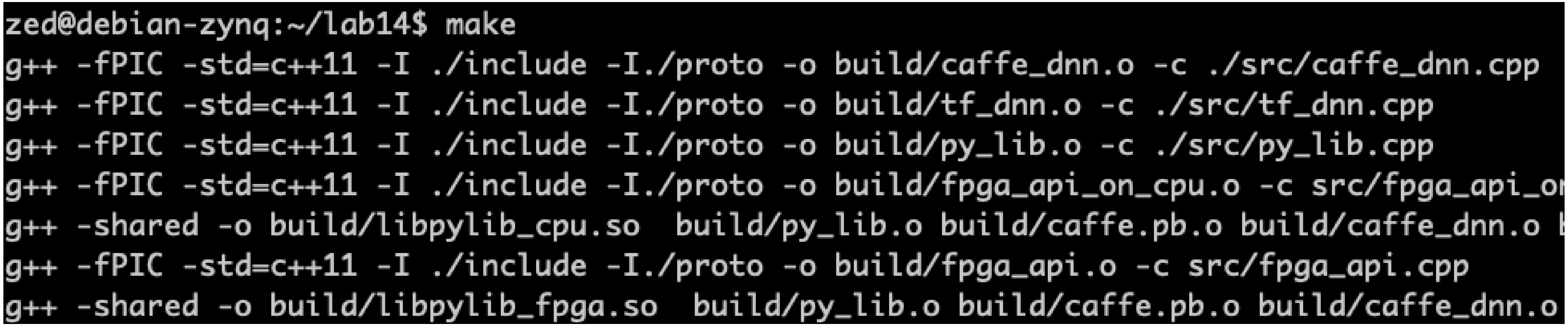
- [Note] Fill other functions `convLowering` etc…

Example: Download datasets



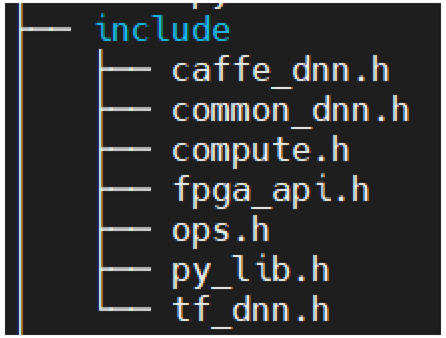
```
zed@debian-zynq:~/lab14$ bash download.sh
converted 'https://dl.dropbox.com/s/mdwy0kzf57nfl5f/t10k-images.idx3-ubyte' (ANSI_X3.4-1968) -> 'ht
--2019-06-03 22:17:32--  https://dl.dropbox.com/s/mdwy0kzf57nfl5f/t10k-images.idx3-ubyte
Resolving dl.dropbox.com (dl.dropbox.com)... 162.125.80.6, 2620:100:6030:6::a27d:5006
Connecting to dl.dropbox.com (dl.dropbox.com)|162.125.80.6|:443... connected.
HTTP request sent, awaiting response... 302 FOUND
Location: https://dl.dropboxusercontent.com/s/mdwy0kzf57nfl5f/t10k-images.idx3-ubyte [following]
converted 'https://dl.dropboxusercontent.com/s/mdwy0kzf57nfl5f/t10k-images.idx3-ubyte' (ANSI_X3.4-1
--2019-06-03 22:17:32--  https://dl.dropboxusercontent.com/s/mdwy0kzf57nfl5f/t10k-images.idx3-ubyte
Resolving dl.dropboxusercontent.com (dl.dropboxusercontent.com)... 162.125.80.6, 2620:100:6030:6::a
Connecting to dl.dropboxusercontent.com (dl.dropboxusercontent.com)|162.125.80.6|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7840016 (7.5M) [application/octet-stream]
Saving to: 'data/t10k-images.idx3-ubyte'
```

Example: Build your implementations



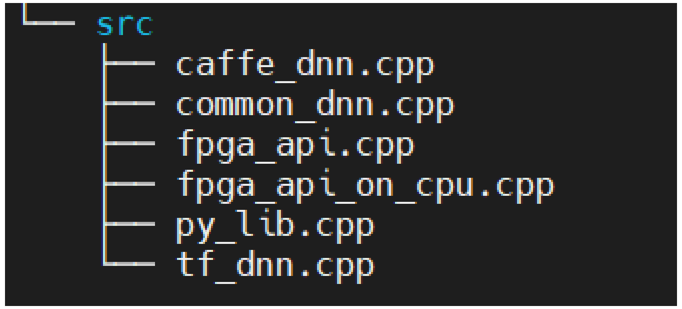
```
zed@debian-zynq:~/lab14$ make
g++ -fPIC -std=c++11 -I ./include -I./proto -o build/caffe_dnn.o -c ./src/caffe_dnn.cpp
g++ -fPIC -std=c++11 -I ./include -I./proto -o build/tf_dnn.o -c ./src/tf_dnn.cpp
g++ -fPIC -std=c++11 -I ./include -I./proto -o build/py_lib.o -c ./src/py_lib.cpp
g++ -fPIC -std=c++11 -I ./include -I./proto -o build/fpga_api_on_cpu.o -c src/fpga_api_o
g++ -shared -o build/libpylib_cpu.so  build/py_lib.o build/caffe.pb.o build/caffe_dnn.o b
g++ -fPIC -std=c++11 -I ./include -I./proto -o build/fpga_api.o -c src/fpga_api.cpp
g++ -shared -o build/libpylib_fpga.so  build/py_lib.o build/caffe.pb.o build/caffe_dnn.o
```

Skeleton Code



```
├── include
│   ├── caffe_dnn.h
│   ├── common_dnn.h
│   ├── compute.h
│   ├── fpga_api.h
│   ├── ops.h
│   ├── py_lib.h
│   └── tf_dnn.h
```

- Edit if you need -> fpga_api.h



```
└── src
    ├── caffe_dnn.cpp
    ├── common_dnn.cpp
    ├── fpga_api.cpp
    ├── fpga_api_on_cpu.cpp
    ├── py_lib.cpp
    └── tf_dnn.cpp
```

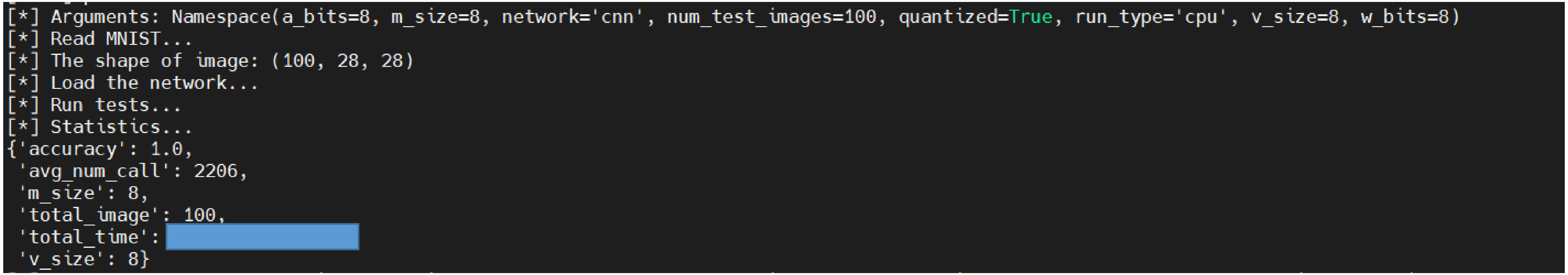
- Edit -> fpga_api.cpp
- Edit -> fpga_api_cpu.cpp

Checklists

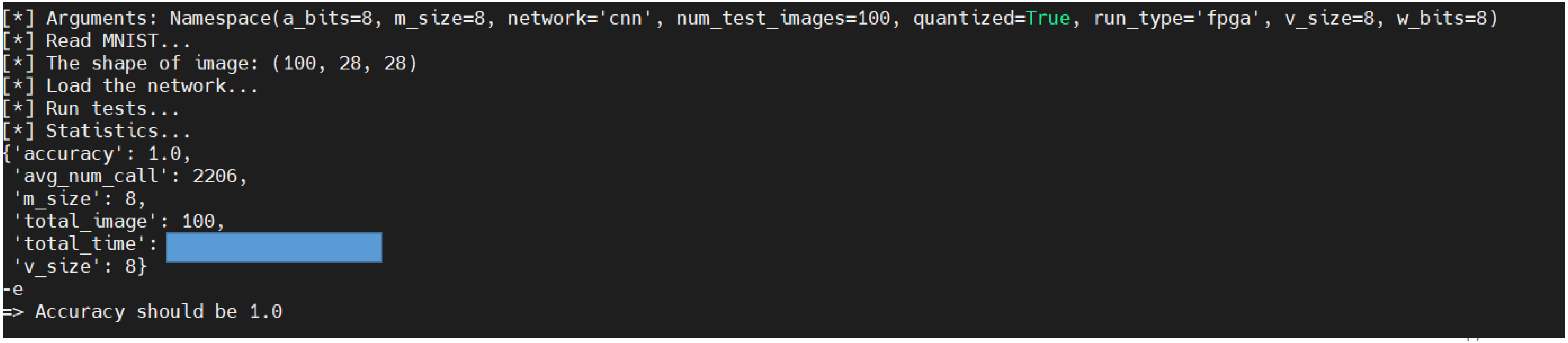
- Convolution Lowering
  - $ sudo python eval.py --num_test_images 100 --m_size 8 --v_size 8 **--network cnn** --run_type [cpu|fpga]
- **Quantization**
  - $ sudo python eval.py --num_test_images 100 --m_size 8 --v_size 8 **--** network cnn --run_type [cpu|fpga] **--quantized**

Example: Benchmark

- $ sudo bash benchmark.sh



```
[*] Arguments: Namespace(a_bits=8, m_size=8, network='cnn', num_test_images=100, quantized=True, run_type='cpu', v_size=8, w_bits=8)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 2206,
 'm_size': 8,
 'total_image': 100,
 'total_time': [redacted]
 'v_size': 8}
```



```
[*] Arguments: Namespace(a_bits=8, m_size=8, network='cnn', num_test_images=100, quantized=True, run_type='fpga', v_size=8, w_bits=8)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 2206,
 'm_size': 8,
 'total_image': 100,
 'total_time': [redacted]
 'v_size': 8}
-e
=> Accuracy should be 1.0
```

Optional Project - Quantization

This is an optional project that you can challenge if you are interested in improving performance

Optimize your MM

- Enable quantization working on CPU / FPGA
  - Quantization on CPU: Implement on fpga_api_cpu.cpp
  - Quantization on FPGA: Implement on fpga_api.cpp

- Verilog code -> Int8 MM
  - Your data is quantized into 8bit, and it is transferred to your bitstream.
  - You don’t need to use IP catalog, just implement the Int 8bit MM.

- If you are curious or confused about something,
  Just question about it!