Binary Neural Network

Inference using tflite with larq

Fatou Kiné SOW
Green AI UPPA

May 31, 2022

Overview

1. Introduction
2. Motivation
3. BNN inference engine
4. Larq Compute Engine
5. Binary Convolution on GPU

Introduction

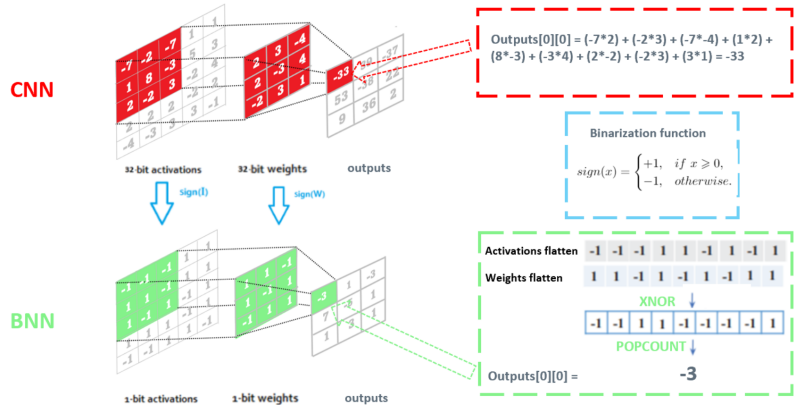


Figure 1: source (A comprehensive review of Binary Neural Network)

Reduce memory consumption and calculation complexity

Motivations

- No storage of parameters in binary format
- Impossible to use XNOR and POPCOUNT operators

Inference Engine

Several BNN inference frameworks are available in open source for optimal use of BNN :

- BMXNet and BMXNet2 (Binary Mix-Net)
- daDNN (Domain-Aware Deep Neural Network)
- FINN
- Larq, the fastest BNN framework in the state of the art compared to existing inference frameworks

Larq Compute Engine

An open-source inference engine for BNN optimization.

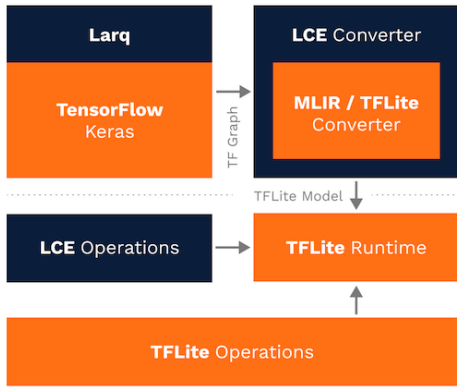


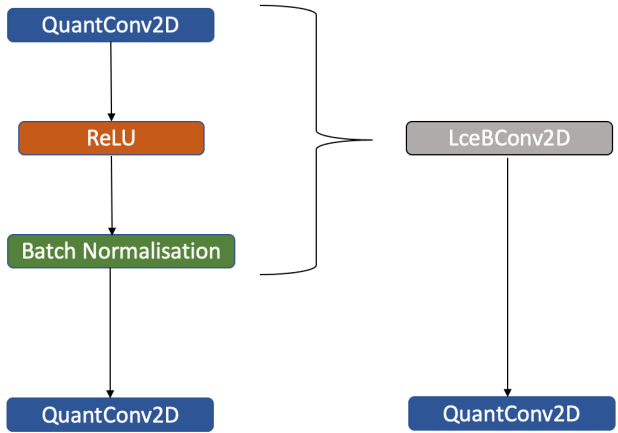
Figure 2: source (A comprehensive review of Binary Neural Network)

Larq Compute Engine

How does LCE optimize inference?

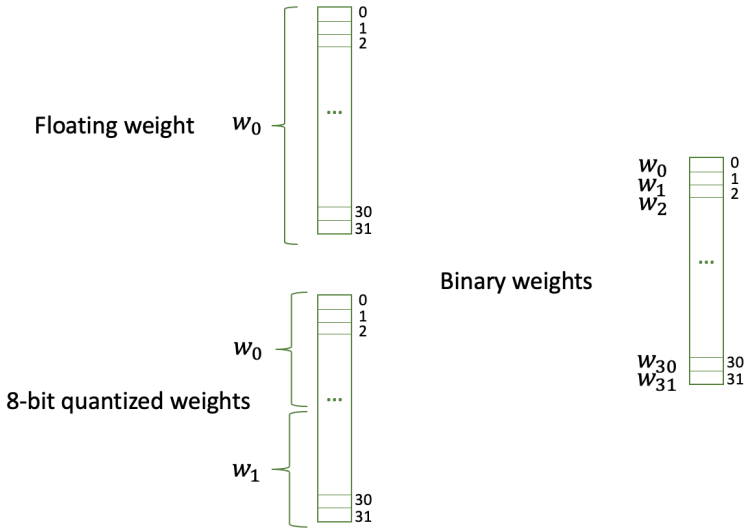
1. MLIR (Multi-Level Intermediate Representation) to optimize graphs by merging some operations
2. Storing weights in binary format
3. Binary matrix multiplication with a BGEMM kernel (Binary GEneral Matrix Multiplication)
   - Tiling to maximize the number of cache hits
   - Vectorization (SIMD) to maximize the computational throughput

Graph optimization



The LCE converter performs these kinds of advanced optimizations automatically, without changes to training code or instruction from the user.

Binary weight storage



Tiling

An optimization technique to maximize the number of cache hits.

- Compute output 4x4 matrix C with two inputs 4x4 matrix A and B
- 16 threads organized into a 2x2 block and 4 blocks in a grid
- Each thread of a block can see what the three other threads of the block have shared

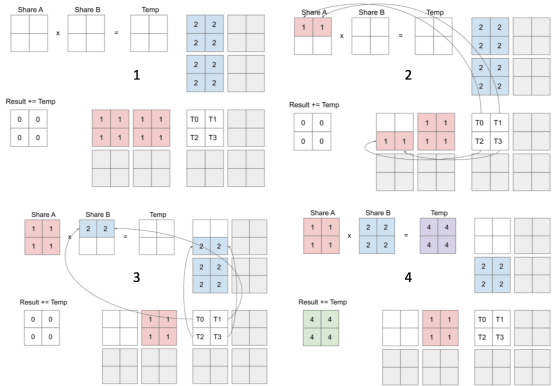


Figure 3: source https://penny-xu.github.io/blog/tiled-matrix-multiplication

Tiling

- Store the intermediate result
- Add it with the result of the next multiplication
- Each thread would load its corresponding result in the output C element

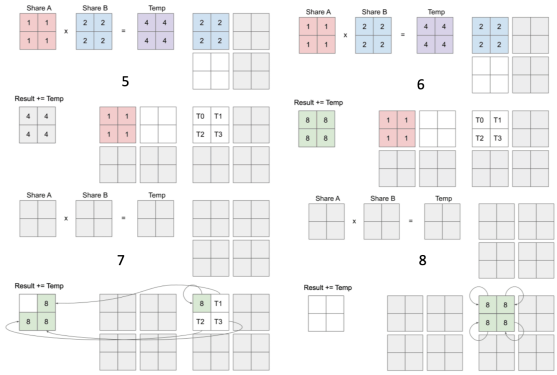
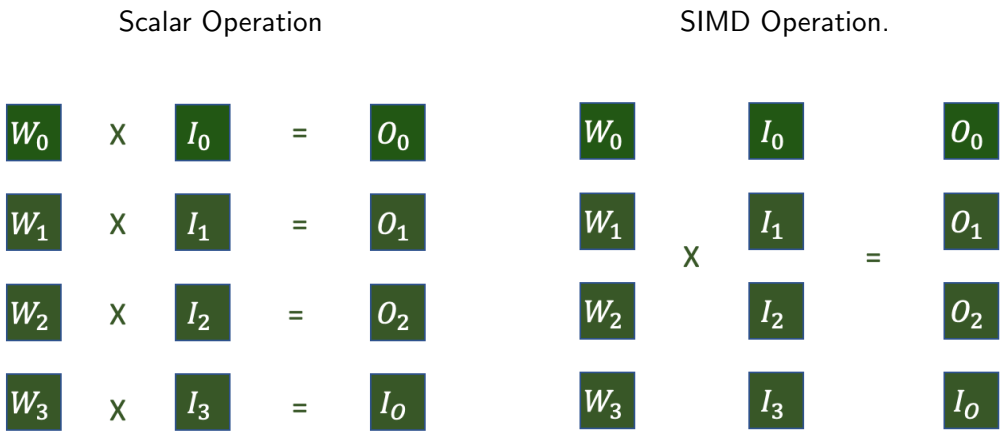


Figure 4: source https://penny-xu.github.io/blog/tiled-matrix-multiplication

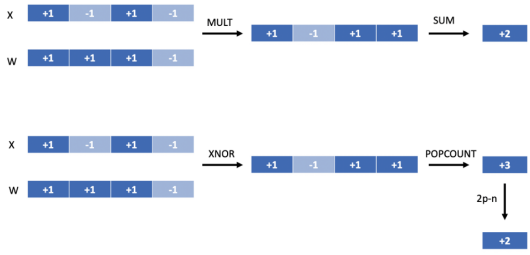
Without tiling, each thread has 8 memory accesses whereas with a tiling, they each have 4 memory accesses.

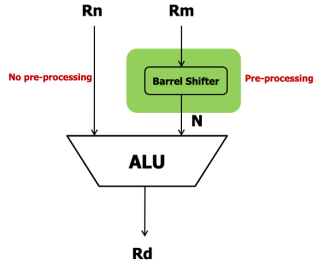
Vectorization

Vectorization to maximize the computational throughput



Binary Matrix Multiplication





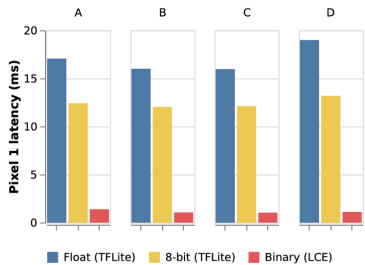
- xnor : *eor* $< Rd >, < Rn >, < Rm >$
- popcount : *cnt* $< Rd >, < Rn >$
- *addp* and *uadalp* to combine n-bit results into 2N-bit results.

BGEMM kernel

```
33  #define XOR_POPCOUNT_ACCUM_LOAD_BLOCK_128      \
34    "eor v0.16b, %[a_0].16b, %[w_0].16b\n\t"     \
35    "eor v1.16b, %[a_1].16b, %[w_0].16b\n\t"     \
36    "eor v2.16b, %[a_2].16b, %[w_0].16b\n\t"     \
37    "eor v3.16b, %[a_3].16b, %[w_0].16b\n\t"     \
38    "eor v4.16b, %[a_0].16b, %[w_1].16b\n\t"     \
39    "eor v5.16b, %[a_1].16b, %[w_1].16b\n\t"     \
40    "eor v6.16b, %[a_2].16b, %[w_1].16b\n\t"     \
41    "eor v7.16b, %[a_3].16b, %[w_1].16b\n\t"     \
42    "eor v8.16b, %[a_0].16b, %[w_2].16b\n\t"     \
43    "eor v9.16b, %[a_1].16b, %[w_2].16b\n\t"     \
44    "eor v10.16b, %[a_2].16b, %[w_2].16b\n\t"    \
45    "eor v11.16b, %[a_3].16b, %[w_2].16b\n\t"    \
46    "eor v12.16b, %[a_0].16b, %[w_3].16b\n\t"    \
47    "eor v13.16b, %[a_1].16b, %[w_3].16b\n\t"    \
48    "eor v14.16b, %[a_2].16b, %[w_3].16b\n\t"    \
49    "eor v15.16b, %[a_3].16b, %[w_3].16b\n\t"    \
50    "cnt v0.16b, v0.16b\n\t"                     \
51    "cnt v1.16b, v1.16b\n\t"                     \
52    "cnt v2.16b, v2.16b\n\t"                     \
53    "cnt v3.16b, v3.16b\n\t"                     \
54    "cnt v4.16b, v4.16b\n\t"                     \
55    "cnt v5.16b, v5.16b\n\t"                     \
56    "cnt v6.16b, v6.16b\n\t"                     \
57    "cnt v7.16b, v7.16b\n\t"                     \
58    "cnt v8.16b, v8.16b\n\t"                     \
59    "cnt v9.16b, v9.16b\n\t"                     \
60    "cnt v10.16b, v10.16b\n\t"                   \
61    "cnt v11.16b, v11.16b\n\t"                   \
62    "cnt v12.16b, v12.16b\n\t"                   \
63    "cnt v13.16b, v13.16b\n\t"                   \
64    "cnt v14.16b, v14.16b\n\t"                   \
65    "cnt v15.16b, v15.16b\n\t"                   \
66    "addp v0.16b, v0.16b, v4.16b\n\t"            \
67    "addp v1.16b, v1.16b, v5.16b\n\t"            \
68    "addp v2.16b, v2.16b, v6.16b\n\t"            \
69    "addp v3.16b, v3.16b, v7.16b\n\t"            \
70    "addp v8.16b, v8.16b, v12.16b\n\t"           \
71    "addp v9.16b, v9.16b, v13.16b\n\t"           \
72    "addp v10.16b, v10.16b, v14.16b\n\t"         \
73    "addp v11.16b, v11.16b, v15.16b\n\t"         \
74                                                 \
```

Gain with Larq Compute Engine

Precision	MAC instruction sequence	Throughput (instructions / cycle)	Throughput (MACs / cycle)
Float	`fmla`	2	8
8-bit	`sdot`	2	32
Binary	`eor`	2	78
	`cnt`	1	
	`addp` / `uadalp`	2 / 1	



In terms of height × width × in channels × out channels, the convolutions are (A) 56×56×64×64; (B) 28×28×128×128; (C) 14×14×256×256; (D) 7×7×256×256.

Binary Convolution on GPU

Optimization of XNOR Convolution for Binary Convolutional Neural Networks on GPU [Mete C. Kaya, Alperen Inci, Alptekin Temizel , July 2020]

- Implementation of binary convolutional network inference on GPU by focusing on optimization of XNOR convolution
- Speed-up of up to 42.61$\times$ with a kernel size of 3$\times$3 and XNOR-Net binary network as reference method on Nvidia GTX1080TI GPU

Binary neural networks for speech recognition [Yan-min QIAN, Xu XIANG, 2019]
Fast implementation of binary matrix multiplication on CPU and GPU architectures with a 5–7 times speedup compared with full precision floating-point matrix multiplication

Binary Convolution on GPU

Binary convolution on GPU has the following steps:

1. XNOR Convolution Bit operations
   - Conversion of input data type to binary type
   - XNOR bitwise logical operation on binary data with binary weights
   - Summation of output binary bits where 0 values are considered as -1.
   - Converting Binary to float data type.
2. XNOR Convolution Scaling Factor Computation
   - Channel-wise summation of input data.
   - Multiplication of matrix K with the scalar α value
3. Multiplication of float output of XNOR convolution with K and α values

Binary Convolution on GPU

Algorithm 1 Input Image to Binary Image

```
for j < IMAGE_HEIGHT do
    for i < IMAGE_WIDTH do
        register_image = input_image[j][i] >> Shift
        Shift+ = 1
        if i mod register_x == 7 then
            i = i − (kernel_size_x − 1)/2
        end if
    end for
    if j mod register_y == 7 then
        j = j − (kernel_size_y − 1)/2
    end if
end for
```

Algorithm 2 XNOR Convolution

```
Mask = weight_matrix_of_ones
iteration_row = register_size_x − kernel_size_x + 1
iteration_col = register_size_y − kernel_size_y + 1
for j < iteration_row do
    for i < iteration_col do
        register_image = (input_image[j][i] >> Shift) ⊕ Weight_Kernel
        register_image = register_image ∧ Mask
        Shift+ = 1
    end for
end for
```

Experimental Evaluation on GPU

Input Size	Vanilla Conv.	XNOR Conv.	Speed-up
256×256	0.062	0.024	$2.57\times$
512×512	0.186	0.069	$2.69\times$
1024×1024	0.671	0.252	$2.66\times$
2048×2048	2.641	0.986	$2.68\times$

Table 1: Comparison of vanilla convolution with XNOR convolution on Nvidia GTX1080TI GPU (ms)

Experimental Evaluation on GPU

Input Size	CPU	GPU	Speed-up
256×256	3.437	0.061	$56.34\times$
512×512	10.623	0.186	$57.11\times$
1024×1024	35.811	0.671	$53.37\times$
2048×2048	132.714	2.641	$50.25\times$

Table 2: Comparison of Intel i7700 CPU and Nvidia GTX1080TI GPU performance for vanilla convolution (ms)

Input Size	CPU	GPU	Speed-up
256×256	0.743	0.0237	$31.35\times$
512×512	2.531	0.0692	$36.57\times$
1024×1024	10.088	0.2519	$40.04\times$
2048×2048	42.011	0.9859	$42.61\times$

Table 3: Comparison of CPU and GPU performance for XNOR convolution (ms).

References


T.Bannink and A. Bakhtiari and A. Hillier and L. Geiger and T. de Bruin and L. Overweel (2020)
Larq Compute Engine: Design, Benchmark, and Deploy State-of-the-Art Binarized Neural Networks
Journal ArXiv abs/2011.09398.


Yuan, Chunyu and Agaian, Sos (2021)
A comprehensive review of Binary Neural Network
https://doi.org/10.48550/arXiv.2110.06804


Mete C. Kaya, Alperen Inci, Alptekin Temizel (July 2020)
Optimization of XNOR Convolution for Binary Convolutional Neural Networks on GPU
https://doi.org/10.48550/arXiv.2007.14178


Xiang, Xu and Qian, Yanmin and Yu, Kai (2017)
Binary Deep Neural Networks for Speech Recognition
doi 10.21437/Interspeech.2017-134


Penny Xu (2019)
Tiled Matrix Multiplication
https://penny-xu.github.io/blog/tiled-matrix-multiplication