XNOROP: a New Metric for Binary Neural Networks Performance Measurement

Ali Shakkouf
ITMO University
Saint Petersburg, Russia
ashakkuf@itmo.ru

Gromov Vladislav Sergeevich
ITMO University
Saint Petersburg, Russia
gromov@itmo.ru

***Abstract*—The quantity of multiply–accumulate operations in a model (MACs) is a widely used metric in the field of neural networks and deep learning. This metric expresses the computational cost of a considered model. However, in Binary Neural Networks BNNs, we merely have floating point operations at inference time, we have mostly XNOR binary operations, hence we can't use the metric MACs to describe the computation cost of BNNs. In this paper, we propose XNOROPs; a new metric that expresses the computation cost of BNNs on Central Processing Units CPUs and Microcontrollers unit MCUs. A compression technique for BNNs is introduced to maximize the usage of CPU and MCU resources, and so reduce the inference time. The new metric is well explained and built-up step by step taking into consideration the inner operations cost in terms of CPU and MCU cycles. XNOROP is related to the well-known MAC metric by a mathematical equation, enabling us to measure the number of operations in a binarized model when number of operations in its float counterpart is provided. Finally, a recipe that helps is choosing the appropriate hardware for BNNs deployment using the new XNOROP metric is provided.**

I. Introduction

When evaluating the computational complexity of AI models, particularly machine learning and deep learning models, several key metrics can be used to assess and compare their efficiency. These metrics focus on different aspects of the model's resource usage, such as time, space, and the number of operations involved. In the AI community there are three well-known metrics to calculate the number of operations involved in a model, or as a measurement of hardware performance. Those metrics are called FLOPS, FLOPs and MACs.

FLOPS [1], [2], [3] (with all uppercase is the abbreviation) stands for floating point operations per second, is a metric invented by [1] in 1979 which refers to the computation speed and is generally used as a measurement of hardware performance (computation capability); like what NVIDIA provides in the characteristics of its' GPUs. FLOPs (lowercase "s" stands for plural) on the other hand, refers to the quantity of floating-point operations in a model. It is commonly used to calculate the computation complexity of an algorithm or model. Also, it is a widely used metric that AI researchers use when they make improvement in some tasks that goes beyond the state of the art in some field [4], [5], [6]. MAC [3], [7] stands for multiply–accumulate operation. MAC represents two operations, multiplication of two numbers and adding that product to an accumulator. For example, when working with different models' architectures such as DenseNet [8] or MobileNet [9] or for edge devices, people use MACs or FLOPs to estimate the model performance.

The reason we use the word "estimate" is that both metrics are approximations instead of the actual capture of the runtime performance. However, they still can provide very useful insights on energy consumption or computational requirements, which is quite useful in edge computing. The AI community uses both terms in their research and scientific papers, with the known assumption that one MAC equals roughly two FLOPs.

In BNNs we do not need to perform multiplication [10], which is a computationally expensive operation especially for microcontrollers that don't have Floating Point Unit FPU [11], [12]. Bitwise operations are sufficient for this task. Specifically, we need XNOR and bits count. Knowing those facts about BNNs, we understand that MACs and FLOPs are not suitable metrics to calculate computational cost of BNNs.

BNNs are a relatively recent development in the field of neural networks and machine learning. It emerged in 2015 and is used in a model called BinaryConnect [13]. The concept of binary neural networks, which involves using binary weights and activations to significantly reduce computational complexity and memory usage, gained significant attention since then.

Till now, there is no known metric that measures the computation cost of BNNs during inference time. Also, there is no metric that expresses the quantity of operations and type of those operations in BNNs. In other words, if we have a BNN model ready to be deployed on some embedded device, we can't use any known metric that measures inference time or execution cost of the model on the target device. The only metric that exists by now is Binary Operations Per Second (BOPS) [14], which basically researchers apply to modern datacenter computing workloads, which often involve a significant proportion of non-floating-point operations. However, for BNNs, no metric can express the computation cost without taking the bus size [15] (core registers size) of the target device into consideration.

From now on, in the main text (except section titles) we will refer to Float Convolution as FC and Binary Convolution as BC. The contributions of this paper are:

- We propose a new approach to compress BNNs and do BC effectively.
- XNOROP: a new metric that expresses the computation cost of BNNs on target CPUs or MCUs, with mathematical relations to calculate it.
- Mathematical approximations of the relationship between our metric XNOROPs and the well-known MACs metric.
- Recommendations for the design of BNNs and its filters sizes to achieve optimal usage of system resources.

The paper is organized as follows. Part 2 clarifies the concept of BNN compression and BC. Part 3 discusses and shows the cost of MAC and XNOROP in terms of CPU cycles and memory access times. In part 4 we show the deployment process of BinaryNet model as a test of our theory on the new metric. Part 5 provides a recipe that helps in choosing the appropriate hardware for a chosen BNN model performance using XNOROP metric.

II. BNN COMPRESSION AND BINARY CONVOLUTION

A. Binary convolution

Let's study a simple example of a convolution operation between two filters of size 3×3. Let's take:

$$f_1=\begin{bmatrix}1&-1&-1\\-1&1&1\\1&-1&1\end{bmatrix}, f_2=\begin{bmatrix}-1&1&-1\\1&-1&-1\\-1&1&1\end{bmatrix}$$

Then the convolution result is:

$$f_1\oplus f_2=1.(-1)+(-1).1+(-1).(-1)+(-1).1+1.(-1)+1.(-1)+\ldots$$
$$1.(-1)+(-1).1+1.1=-5$$

In BNNs, weights that have the value -1 are represented as 0 and weights that have the value 1 are represented as 1. This is because it is physically what we can store in one bit of memory, bit could only be set or reset. We therefore need to find a cost-effective method to do BC.

If we replace each -1 by 0, we will obtain convolution result as:

$$f_1\oplus f_2=1.0+0.1+0.0+0.1+1.0+1.0+1.0+0.1+1.1=1$$

Which is not correct. The correct answer is -5 not $+1$. A closer observation on the previous convolution operation shows us that we obtain $+1$ when we multiply one by one $(+1\times+1)$ or minus one by minus one (-1×-1), where minus one is represented as 0, so $-1\times-1\Leftrightarrow0\times0$. This logical behavior is the XNOR operation, truth table and gate for which are shown in Fig. 1. In other words, we can do BC by doing an XNOR between the two filters, and then counting the number of set bits in the binary representation of XNOR operation result.

What is left for us now is set-bits counting, taking into consideration that 0 is actually a -1 and XNOR is equal to NOT XOR. For our example we have:

$$\sim(100011101\,\hat{}\,010100011)=\sim(110111110)=001000001$$

Where $\sim$ is the logical bitwise NOT operation. Now we count the number of ones and zeros to find the result of convolution as:

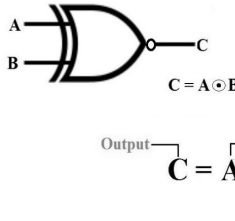
$$f_1\oplus f_2=-1\times NumberOf\,\mathrm{Re}\,setBits+1\times NumberOfSetBits$$
$$NumberOf\,\mathrm{Re}\,setBits=\sum_{i=0}^{f\times f-1}1:b_i=0,b_i\in XNOR(f_1,f_2)$$
$$NumberOfSetBits=\sum_{i=0}^{f\times f-1}1:b_i=1,b_i\in XNOR(f_1,f_2)$$
$$f_1\oplus f_2=-1\times(FilterSize-NumberOfSetBits)+1\times NumberOfSetBits$$
$$f_1\oplus f_2=2\times NumberOfSetBits-FilterSize$$
$$f_1\oplus f_2=2\times\left(\sum_{i=0}^{f\times f-1}1:b_i=1,b_i\in XNOR(f_1,f_2)\right)-f\times f$$

Where b_i is the value of bit i. Let's rewrite the convolution operation:

$$f_1\oplus f_2=2\times\left(\sum_{i=0}^{f\times f-1}1:b_i=1,b_i\in(\sim(100011101\,\hat{}\,010100011))\right)-9$$
$$=2\times\left(\sum_{i=0}^{f\times f-1}1:b_i=1,b_i\in(\sim(110111110))\right)-9=4-9=-5$$

Where 9 is filter size, which in our case is $3\times3=9$, $\hat{}$ is the XOR operation. Therefore, the math behind a BC is:

$$c=f_1\oplus f_2=2\times\left(\sum_{i=0}^{f\times f-1}1:b_i=1,b_i\in XNOR(f_1,f_2)\right)-f\times f \quad (1)$$



Input		Output
A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

Fig. 1. XNOR gate mathematical representation and truth table

B. How to count set bits in binary representation effectively?

The intuitive method to do set-bits counting is by checking the Least Significant Bit (LSB) whether it is 1 or 0, after that we shift the number to the right by 1, and again check the LSB whether it is 1 or 0. This is repeated n times, where n is filter size, as shown in Algorithm 1.

Algorithm 1 set-bits counting by shifting

f_1: first filter.
f_2: second filter.
$xnor$: XNOR (f_1, f_2)
s: filter size
$res\rightarrow0$: convolution result
For i **from** 0 **to** $s-1$ **do**
  If b_i is **equal** to 1 **then**
    $res\rightarrow res+1$
  end
  $xnor\rightarrow$ shift $xnor$ to the right by 1
end
$res\rightarrow2\times res-s$

For most modern processors across different architectures, basic shift operations generally get performed in a single clock cycle. However, the exact number of cycles can depend on factors such as the specific instruction the core can handle, the number of bits being shifted, and the processor's microarchitecture. Since ARM CPUs and MCUs are the most used hardware for embedded systems we will consider it and build our analysis targeting its devices. Performing one XNOR operation takes two CPU cycles, one cycle to do XOR [16] (here we neglect the cycles required to access data from memory and discuss it at a later stage), and the other cycle to do the logic NOT [16] operation. Shift operation takes also one CPU cycle [16], which seems quite fast. However, if BNN has millions of convolution operations, and if data type is unsigned integer 32bits, then for each convolution we have 31 shifting operations, 32 comparisons, 32 add operations and some CPU cycles to access data from memory. This leads us to a situation in which Micro-Controller Units (MCUs) that tick with few hundreds of MHz will merely do one million BCs in one second, and this is very computationally expensive and unsuitable for real time applications.

We introduce a more efficient way to do set-bits counting using lookup tables. The idea is to create a matrix, or a lookup table, in which we store the number of set bits for numbers from 0 to 3, 15, 255, 65535 or 4294967296, depending on how large the free memory is, what it allows us to store in it and core registers sizes. Table I shows an example of lookup table of size 64 Kbytes. The numbers 3, 15, 255, 65535 and 4294967296 are equal to 2^2-1, 2^4-1, 2^8-1, $2^{16}-1$, $2^{32}-1$ respectively. The reason we chose those numbers is because they correspond to the used overtime bus sizes in core architectures of CPUs and MCUs [17], [18], [19], [20]. The Bus size of 64 or 128 bits is not considered, because storing a lookup table of size 2^{64} or 2^{128} in not feasible; requires a huge amount of memory that can't be created.

Table I occupies $65536/1024 = 64$ Kbytes to store in memory. So now when we want to find the number of ones in an unsigned integer 32bits, we simply take the first half of the number (one clock cycle) and lookup for number of ones in the matrix (one access for memory), then take the second half of the number (another clock cycle) and lookup for number of ones in the matrix. And finally, sum up the two numbers together. The operations are one shift, one sum operation and two memory access. So, we moved from more than 64 logical operations in the first solution to 1 operation in the second solution, from 32 sum operation to 1 sum operation and from 64 times of memory access to only two memory accesses. which is quite good optimization.

The idea of doing BC using lookup tables is probably the fastest approach to be used. We did simple modeling that shows how fast we can do set-bits counting using lookup tables approach for tables of sizes 4, 16, 255, 64K Bytes and the slow shifting approach. In this experiment we simulate set-bits counting for numbers from 0 up to 1 million. Modeling results are shown in Fig. 2. It is to be noticed that using lookup table of size 64 Kbytes is $1635m\text{Sec}/61m\text{Sec} = 25.80\times$ faster than counting by shifting. Where $1635m\text{Sec}$ is the time consumed by STM32F429ZI microcontroller to do set-bits counting by shifting, while 61mSec is consumed using lookup table of size 64 Kbytes. Also, we notice that we gain a linear decrease in time for doing the same operations as we increase the size of the table, which as we mentioned before, corresponds to architecture bus and registers sizes [17], [18], [19], [20]. Algorithm 2 demonstrates pseudocode for doing XNOR using lookup table of size 64 Kbytes. The same algorithm is used in case we have lookup table smaller in size. The only difference is the number of shift and add operations, and number of lookup table accesses for one XNOR.

Algorithm 2 set-bits counting using lookup table of size 64 Kbytes and bus size is 32bit

f_1: first filter.
f_2: second filter.
$xnor$: XNOR (f_1, f_2)
$res \rightarrow 0$: convolution result
$res \rightarrow LOOKUP[xnor \,\&\, 0xFFFF]$ // & *is a cast instruction*
$res \rightarrow res + LOOKUP[xnor >> 16]$

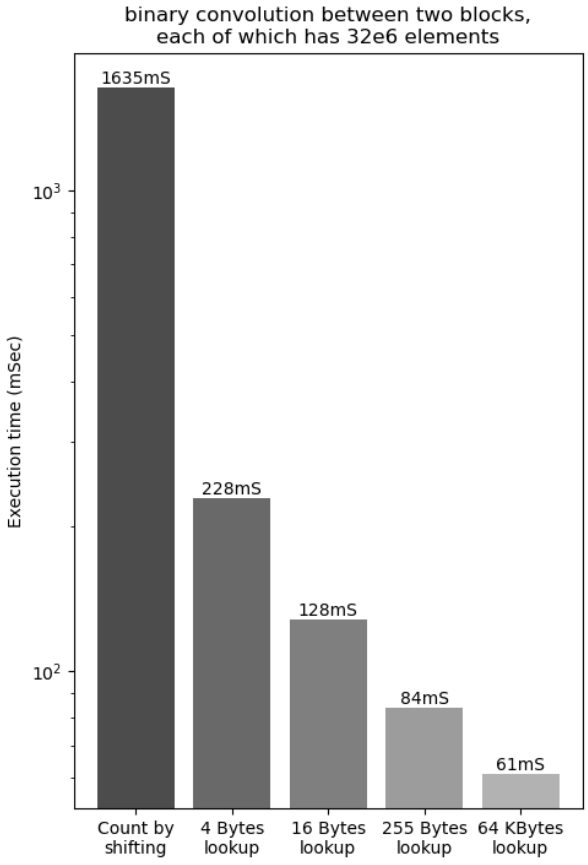


Fig. 2. Set bits counting. Count by shifting is represented in Algorithm 1. Count via lookup table of size 64 Kbytes is represented in Algorithm 2. The latter is $1635m\text{Sec}/61m\text{Sec} = 25.80\times$ faster than the count by shifting.

C. *Binary filters compressing for fast binary convolution*

BC does not just replace MACs operation by XNOR and shift operations, but their quantity will be about 4, 8, 16, 32, 64 or 128 (bus size) times less than the quantity of MACs to calculate the output of a convolution layer of some size. To

demonstrate this idea let's take a convolution layer and proceed in performing BC operation using lookup tables. We study a simple case of two consecutive convolutions blocks as shown in Fig. 3.

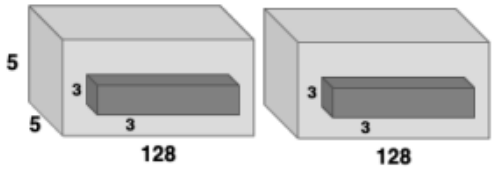


Fig. 3. Two Consecutive conv-layers $64-128\times(3\times3),128-128\times(3\times3)$

Since filter size is $3\times3\times128$ and it is a binary filter; each element of this filter has one of two values 0 or 1, then we can compress this filter (assuming that bus size of target architecture is 32) to be of dimensions $3\times3\times4$ where each element of the compressed filter is 32-bit unsigned integer that represents a chunk of the source filter, size of which is $1\times1\times32$. This compression leads to replacement of $3\times3\times128\times5\times5=28800$ MACs with only $3\times3\times4\times5\times5=900$ XNOR operations, 900×2 add operation and 900 shift operations, which is a huge improvement in performance. It is to be noticed that when the hardware doesn't have FPU unit, then using BC boosts the performance to a level of optimization that is several hundred times faster than FC.

We call the combination of shift, add, and XNOR operations required to perform a BC between two parameters as XNOROP. So, we state that XNOROP expresses the simple logic operations required to perform BC between two integer elements. We decided to join the three types of operations in one term, because we would like our new metric to be comparable to MAC, which expresses float addition and multiplication operations [21].

III. COST OF MAC AND XNOROP

To compare MAC and XNOROP operations in terms of clock cycles, we need to consider how each operation is implemented and executed on CPUs and MCUs that have core architectures from ARM and occupied with FPU unit.

Simple floating-point operations like addition and multiplication are comparable in clock cycles to an XNOR operation [22], while complex floating-point operations like division or square root require significantly more clock cycles. Specifically, XNOR takes two clock cycles to get performed. Each of Adding two integers and shifting operations takes one clock cycle. Float addition and multiplication takes two clock cycles for each operation to get performed. In FC, for each two elements we perform one multiplication (2 clocks) and one addition (2 clocks), so it costs 4 clock cycles. In BC, for each two elements we perform one XNOR (2 clocks), several add operations (1 clock for each) and several shift operations (1 clock for each). XNOROP always requires just one XNOR operation no matter what the bus or lookup table sizes are. What is left for us is to determine exactly the number of add and shift operations involved in one XNOROP.

Suppose that we have a hardware with a bus size B where $B\in\{2,4,8,16,32,64,128\}$, and a lookup table of size S. S obeys the inequality $S\le2^B$ because on a hardware with bus B we can't store numbers bigger than 2^B unless we implement a special storing mechanism, which increase cost of XNOROP. If $S=2^B$ then we can do set-bits counting only one add operation and one access to the created lookup table. If $S=2^{B/d}$, where $d\in\{1,2,4,...,B\}$, then we would need d add operations and $d-1$ shift operation to perform one XNOROP. Table II shows the number of shift and add operations required to do one XNOROP for different bus and lookup table sizes. Cases that don't match the condition of $S\le2^B$ are ignored and represented as "–" in table II. Last wo columns of table II are shaded in gray because by now there is no mechanism that allows us to store lookup tables of such sizes. We indeed need to model these different sizes of lookup table for different bus sizes because the resources on a chosen target hardware on which we might deploy our BNN model vary so much.

TABLE II. SHIFT AND ADD OPERATIONS REQUIRED TO PERFORM ONE XNOROP FOR DIFFERENT BUS AND LOOKUP TABLE SIZES

B \ S	2^2	2^4	2^8	2^{16}	2^{32}	2^{64}	2^{128}
2	1	–	–	–	–	–	–
4	3	1	–	–	–	–	–
8	7	3	1	–	–	–	–
16	15	7	3	1	–	–	–
32	31	15	7	3	1	–	–
64	63	31	15	7	3	1	–
128	127	63	31	15	7	3	1

In summary, doing FC operation between two blocks of size one element takes 4 clock cycles, while doing BC takes $2+d+d-1=2d+1$ clock cycles. If we represent the cost of FC as C_{FC} and the cost of BC as C_{BC}, then the relationship between the two operations could be represented as:

$$\left.\begin{aligned} r=C_{BC}/C_{FC}=(2d+1)/4;\\ d\in\{1,2,4,...,B\},B\in\{2,4,8,16,32,64,128\}\end{aligned}\right\} \quad (2)$$

Fig. 4 is a representation of equation 2. As we can see if $d=1$ then BC costs 3 clock cycles. The situation gets worse to a degree at which BC costs $64.25\times$ times more clock cycles than FC. This is the case when we have a bus size of 128 and we choose to create a lookup table of size 1 Byte. In fact, it is very important to keep in mind that each time we do an FC, we do a convolution operation between two filters of size 1 element. While in BC we do a BC between two filters of size B elements (bus size), which is the core idea of compression method. In other words, approximately (without taking memory access into consideration), for a specified time unit, we can do inference using a BNN that has $B/r\times$ times more parameters than the float NN. An example of that, if we have bus size of 32, then B/r goes from $32/((2\times32+1)/4)\approx1.97\times$ up to $32/((2\times16+1)/4)=42.67\times$ more parameters in BNN model.

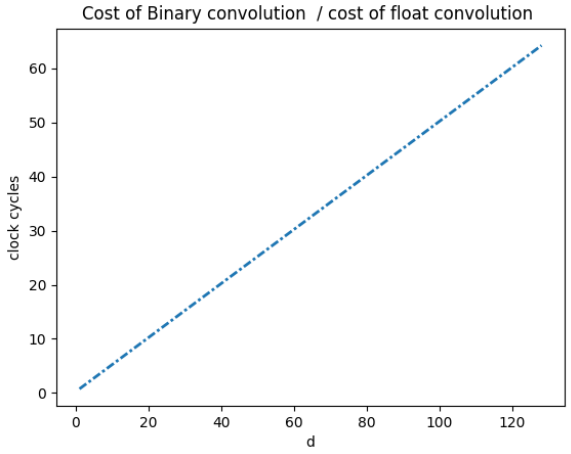


Fig. 4. Relationship between execution cost of BC and FC in terms of clock cycles for different sizes of lookup table.

This compression method enables us to write a mathematical expression that expresses the quantity of XNOROPs operation in a convolution layer as:

$$XNOROPs = s_i \times s_j \times \lceil s_k / B \rceil \times O_i \times O_j \times O_k \quad (3)$$

Where s_i, s_j, s_k are filter dimensions s_k of which corresponds to the number of input channels, B is bus size, or core registers size, O_i, O_j, O_k are layer output dimensions, $\lceil \ \rceil$ is the ceil function that takes the integer upper bound of a float number. As we mentioned, the term XNOROPs represents the quantity of XNOR, integer adds and shift operations. On the other hand, the quantity of MACs could be expressed by the following equations:

$$MACs = s_i \times s_j \times s_k \times O_i \times O_j \times O_k \quad (4)$$

From equations 3 and 4 we see that for a chosen layer, the quantity of MACs is $s_k / \lceil s_k / B \rceil \times$ times more than the quantity of XNOROPs. We call $s_k / \lceil s_k / B \rceil$ as BNNs compression factor.

There is another important factor that we should consider while handling the performance of BCs operations, which is memory access times. Each memory access can take up to 7 clock cycles depending on the type of memory we are accessing.

It is to be noticed that whether we are using float or BC, we will store filters weights or the output of the previous layer in the same type of memory [23]. So, we can ignore how many clock cycles does it takes to access data and instead focus on the difference between memory access times between BC and FC.

The proposed binary compression method allows us, as we mentioned before, to store B filters elements in one integer. Correspondingly, this reduces memory access times by $B\times$ times. Since filters have $3D$ shapes, where the third dimension might not be divisible by B, then it is more accurate to state that in BC we have memory access times $s_k / \lceil s_k / B \rceil \times$ less than its counterpart in the FC. Fig. 5 shows a plot for the relation $s_k / \lceil s_k / B \rceil$ in different scenarios of a convolution layer that might exist. All lines in that figure show step-like behavior (more noticeable for buses of sizes 64 and 128) and this is caused by the ceil function $\lceil \ \rceil$. Moreover, Fig. 5 indicates two major points:

- For a chosen bus size, we gain the optimal performance (the biggest saving in operations) when we choose a filter size that is divisible by bus size, and this optimal saving in quantity of XNOROPs and memory access times equals to bus size of the target architecture. For example, take a filter of size 128 and a bus of size 64, then from Fig. 5 we can tell that saving in operations is 64x times, which is equal to bus size. So, it is an important recommendation while building the architecture of BNN to choose filters sizes to be divisible by the target platform bus size, on which the model will be deployed.
- For a chosen model, the higher the bus size of the target platform the greater savings we get in both XNOROPs and memory access times. So, it is recommendable to pick a platform with a bus size as large as possible.

Let's list the key results that we have obtained by now:

- BC operation is simplified and then expressed as shown in equations 1.
- Set-bits counting is done using lookup table as expressed in algorithm 2, where the bigger the lookup table is, the faster we can count set-bits in the integer that results from XNOR operation.
- The proposed binary filters compression method saves $s_k / \lceil s_k / B \rceil \times$ memory access times less than its float counterpart and requires a quantity of XNOROPs that is $s_k / \lceil s_k / B \rceil \times$ times less that quantity of MACs in its float counterpart.
- ARM CPUs and MCUs might perform XNOROP operation faster or slower than MAC operation as shown in equation 2.
- Doing XNOROP convolution at some layer takes $r \times s_k^2 / \left(\lceil s_k / B \rceil\right)^2 \times$ less clock cycles than FC, where: $r \times s_k^2 / \left(\lceil s_k / B \rceil\right)^2 = r \times \underbrace{s_k / \lceil s_k / B \rceil}_{a} \times \underbrace{s_k / \lceil s_k / B \rceil}_{b}$. Here r represents the relationship as in equation 2, a is compression factor and b comes from memory access times reduction.

IV. EXPERIMENT

To test our theory, we chose the BinaryNet model as an example and used it to classify the CIFAR10 dataset. We trained the model for 75 epochs then deployed it on the STM32F429ZI microcontroller. STM32F429ZI ticks at a frequency 180MHz. The microcontroller has bus size of 32 bit. In this experiment we use lookup table of size 64 Kbytes. BinaryNet is a sequential model, in which layers are stacked one after the other. Each image in CIFAR10 has a size of $32 \times 32 \times 3$. If we refer to Float Convolution layer as FC,

Binary Convolution layer as BC, Batch Normalization layer as BN, Max Pooling layer as MP, Binary Dense layer as BD, then our model is constructed of the following layers in order:

FC1(32, 3) → BN2() → BC3(128, 3) → MP4(2,2) → BN5() → BC6(256, 3) → BN7() → BC8(256, 3) → MP9(2,2) → BN10() → BC11(512, 3) → BN12() → BC13(512, 3) → MP14(2,2) → BN15() → Flatten16 → BD17(1024) → BN18() → BD19(1024) → BN20() → BD21(10) → BN22() → Softmax23. The number after each layer represents its order in the sequential model, starting from 1 for FC layer and ending with 23 for SoftMax.

Binarizing the first and last layers hurts accuracy much more than binarizing other layers in the network. Meanwhile, the number of weights and operations in these layers are often relatively small. Therefore, it has become standard to leave these layers in higher precision [24].

BCs and BDs layers take most of clock cycles required to do an inference in every model we might use, so we will analyze the systems -for simplicity- ignoring batch normalization and max pooling layers.

The layer BC3(128, 3) has 128 filters, each of which has a size of $30\times30\times32$ and outputs a feature map of size $30\times30\times128$. Using equation 3 we can find that the number of XNOROPs in this layer is $3\times3\times\lceil32/32\rceil\times30\times30\times128=1036k$. Using the same equation, we find the number of XNOROPs in all BCs and BDs layers as shown in Table III.

TABLE III. NUMBER OF XNOROPS AND MAC IN FC, BC AND DC LAYERS OF BINARYNET MODEL

Laper	XNOROPs	MACs
FC1(32, 3)	-	1555k
BC3(128, 3)	1036k	-
BC6(256, 3)	2073k	-
BC8(256, 3)	4147k	-
BC11(512, 3)	1806k	-
BC13(512, 3)	3612k	-
BD17(1024)	147k	-
BD19(1024)	32k	-
BD21(10)	1k	-
Sum	12854k	1555k

Each MAC operation takes 4 clock cycles as mentioned before, but we add to it one more clock cycle to access weights from flash memory. While each XNOROP consumes 5 clock cycles using equation 2, but also, we add to it one more clock cycle to access weights from flash. Knowing that, we can find the clock cycles required to do one inference over our model as:

$$12854\times6+1555\times5=84899k \quad clock\ cycles$$

So theoretically, on our target microcontroller, our model should run in about $85/180\approx0.472\text{Sec}$, where $180\Leftrightarrow180MHz$ which is controller ticks per second. We implemented the inference code on our controller, and it took 0.511Sec to do an inference.

The $0.511-0.472=39m\text{Sec}$ is the difference in execution time between theory and practice. This difference is consumed on doing batch normalization, max pooling, binarizing the input for each binary layer and timing purposes for system timer which requires context switch at each interrupt. So, we can say empirically it is better to add about 10% to the theoretical estimation of inference time to stay in the safe area. It is important to notice that XNOROP is an estimation of system performance, not the exact performance. The same fact stands for FLOP and MAC as stated in the introduction.

V. USAGE OF XNOROP TO DETERMINE THE REQUIRED HARDWARE

The XNOROP metric allows us to determine the appropriate hardware for our system. In other words, XNOROP answers the following question: if we have a model M and we would like our model to run at X fps or X inference times in a second, what is the necessary hardware that offers such a performance. Here we introduce the recipe step by step to choose the target hardware:

- Build up table that represents number of MACs and XNOROPs in the model as shown in table III.
- Calculate the required clock cycles to do one inference using equations 2 and 3, then increase the result by 10% to stay in safe area.
- Take the result obtained in the previous step and multiply it by the number of inference times needed in one second to get the minimum required clock rate. Choose a target hardware that offers a clock rate equal or bigger than the required clock rate.

VI. CONCLUSION

The deployment of BNNs on tiny devices and embedded systems is a very interesting and important problem to study. The XNOROP metric is a key metric that helps in BNN analyzing and simplifies the deployment process over embedded devices (CPUs and MCUs). The drawback of this metric is that we need to perform calculation over all layers manually. There is no API in PyTorch or TensorFlow that helps in computing the XNOROP for a chosen model, like what exists for FLOP and MAC. We hope that our proposed metric will be implemented in the upcoming version of those very well-known AI frameworks.

Although BNNs have some aspects to be used in, a few challenges and constraints remain an open issue for research. In this paper, we introduced a new metric that expresses the complexity of BNNs and number of operations in a BNN. We introduced a compression method and with it a compression factor that measures the resources that we save using BNNs compared to its float counterpart.

ACKNOWLEDGMENT

We would like to thank the team-lead in intelligent roads company and the administration, Dean and Dean assistant of the control systems and robotics for their support and the big facilities they provided to get this research done.

TABLE I. LOOKUP TABLE OF SIZE 64 KBYTES

Number	0	1	2	3	4	…	65531	65532	65533	65534	65535
Number of ones	0	1	1	2	1		14	14	15	15	16

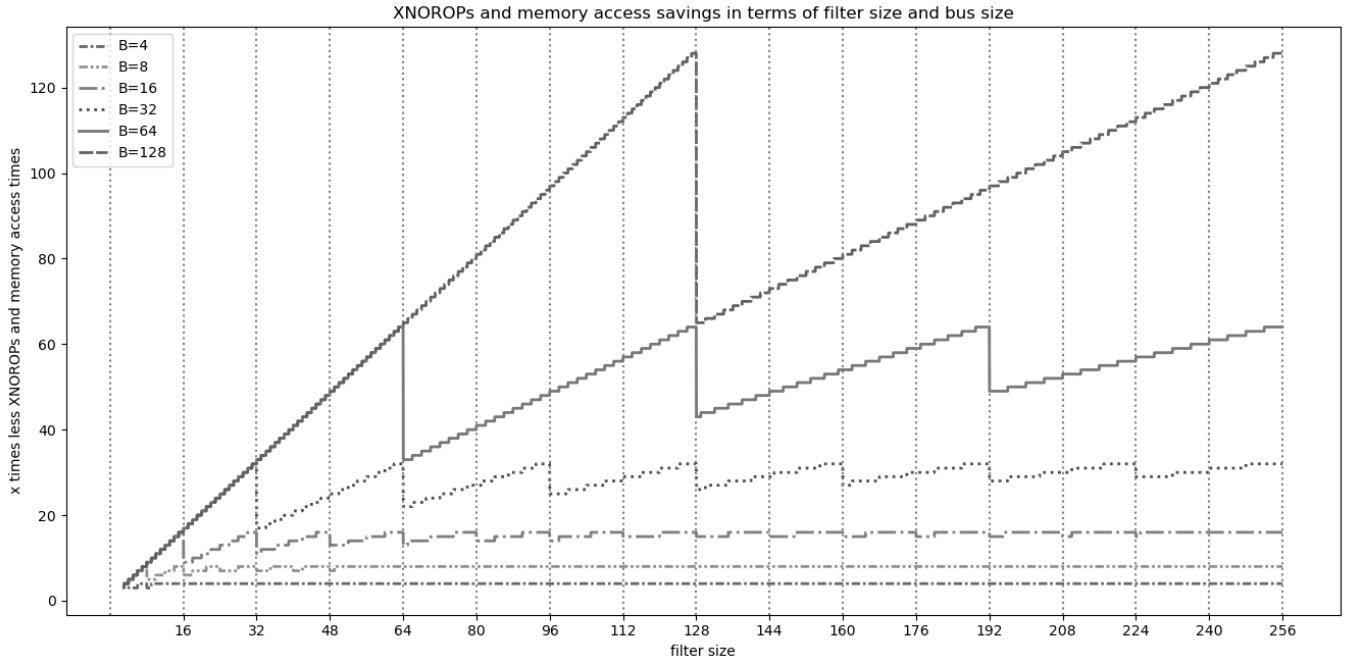


Fig. 5. Compression factor $s_k / \lceil s_k / B \rceil$ in different scenarios of a convolution layer that might exist. The y access expresses the number of memory access times and the factor at which we reduce the XNOROPs with comparison to MACs. The vertical dotted lines were drawn to show at what filter sizes we gain maximum reduction in resources usage.

REFERENCES

[1] Dongarra, J. (1979). Performance of various computers using standard linear equations software in a Fortran environment. Argonne National Laboratory Report ANL-80-70, Argonne, Illinois.

[2] FLOPS." Wikipedia, The Free Encyclopedia, Wikimedia, 11 July 2024, https://en.wikipedia.org/wiki/FLOPS

[3] "Calculate Computational Efficiency of Deep Learning Models with FLOPs and MACs." KDnuggets, June 19, 2023, https://www.kdnuggets.com/2023/06/calculate-computational-efficiency-deep-learning-models-flops-macs.html

[4] Tan, S., Zhang, Z., Cai, Y., Ergu, D., Wu, L., Hu, B., Yu, P., & Zhao, Y. (2024). SegStitch: Multidimensional Transformer for Robust and Efficient Medical Imaging Segmentation. https://arxiv.org/abs/2408.00496

[5] Lin, X. V., Shrivastava, A., Luo, L., Iyer, S., Lewis, M., Gosh, G., Zettlemoyer, L., & Aghajanyan, A. (2024). MoMa: Efficient Early-Fusion Pre-training with Mixture of Modality-Aware Experts. https://arxiv.org/abs/2407.21770

[6] Cui, Z., Yao, J., Zeng, L., Yang, J., Liu, W., & Wang, X. (2024). LKCell: Efficient Cell Nuclei Instance Segmentation with Large Convolution Kernels. https://arxiv.org/abs/2407.18054

[7] "Multiply–accumulate operation." Wikipedia, The Free Encyclopedia, Wikimedia Foundation, 12 July 2024, https://en.wikipedia.org/wiki/Multiply–accumulate_operation

[8] Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2018). Densely Connected Convolutional Networks. https://arxiv.org/abs/1608.06993

[9] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. https://arxiv.org/abs/1704.04861.

[10] Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016). XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. https://arxiv.org/abs/1603.05279.

[11] ARM Holdings. (2014). ARM Cortex-M4 processor technical reference manual. Retrieved from https://developer.arm.com

[12] Texas Instruments. (2005). TMS320C2000 series technical reference manual. Retrieved from https://www.ti.com

[13] Courbariaux, M., Bengio, Y., & David, J.-P. (2016). BinaryConnect: Training Deep Neural Networks with binary weights during propagations. https://arxiv.org/abs/1511.00363

[14] Wang, L., Zhan, J., Gao, W., Yang, K., Jiang, Z., Ren, R., He, X., & Luo, C. (2019). BOPS, Not FLOPS! A New Metric and Roofline Performance Model For Datacenter Computing. https://arxiv.org/abs/1801.09212

[15] ARM Holdings. (2021). ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile. Retrieved from ARM Developer

[16] ARM Holdings. (2014). *ARM Cortex-M4 processor technical reference manual* (Rev. r0p1). Retrieved from https://developer.arm.com/documentation/ddi0439/latest

[17] Intel Corporation. *Intel 4004 Microprocessor Data Sheet*. Intel, 1971. https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/4004-datasheet.pdf.

[18] Microchip Technology Inc. ATmega328P Datasheet. Microchip Technology Inc., 2016, https://www.microchip.com/wwwproducts/en/ATmega328P.

[19] STMicroelectronics. STM32F429ZI Manual. STMicroelectronics, 2020, https://www.st.com/resource/en/reference_manual/dm00031020.pdf.

[20] ARM Limited. ARM Cortex-A76 Reference Manual. ARM Limited, 2018, https://developer.arm.com/documentation/den0024/latest.

[21] Moosmann, J., Bonazzi, P., Li, Y., Bian, S., Mayer, P., Benini, L., & Magno, M. (2023). Ultra-Efficient On-Device Object Detection on AI-Integrated Smart Glasses with TinyissimoYOLO. https://arxiv.org/abs/2311.01057

[22] Engblom, Jakob & Ermedahl, Andreas & Sjdin, M. & Gustafsson, Jan & Hansson, Hans. (2001). Execution-time analysis for embedded

real-time systems. International Journal on Software Tools for Technology Transfer - STTT.

[23] H. Miao and F. X. Lin, "Towards Out-of-core Neural Networks on Microcontrollers," *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, Seattle, WA, USA, 2022, pp. 1-13, doi: 10.1109/SEC54971.2022.00008.

[24] Geiger et al., (2020). Larq: An Open-Source Library for Training Binarized Neural Networks. Journal of Open Source Software, 5(45), 1746, https://doi.org/10.21105/joss.01746