Towards Efficient Quantized Neural Network Inference on Mobile Devices

Yaman Umuroglu, Magnus Jahre
Norwegian University of Science and Technology
Trondheim, Norway
yamanu@ntnu.no

To appear as a work-in-progress paper at the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), October 2017.

1 INTRODUCTION

From voice recognition to object detection, *Deep Neural Networks (DNNs)* are steadily getting better at extracting information from complex raw data. Combined with the popularity of mobile computing and the rise of the Internet-of-Things (IoT), there is enormous potential for widespread deployment of intelligent devices, but a computational challenge remains. A modern DNN can require billions of floating point operations to classify a single image, which is far too costly for energy-constrained mobile devices. Offloading DNNs to powerful servers in the cloud is only a limited solution, as it requires significant energy for data transfer and cannot address applications with low-latency requirements such as augmented reality or navigation for autonomous drones.

Quantized Neural Networks (QNNs) have recently emerged as a potential solution to this problem. They contain convolutional, fully-connected, pooling and normalization layers similar to the floating point variants, but use a constrained set of values to represent each weight and activation in the network. We will use the notation $\mathbf{W^w A^a}$ to refer to a QNN with w-bit weights and a-bit activations, and focus on cases where they represent *few-bit integers* ($w, a \leq 4$). The computational advantages of such QNNs are two-fold:

(1) Each parameter and activation can be represented with a few bits. A greater portion of the working set can thus be kept in on-chip memory, enabling greater performance, reducing off-chip memory accesses and the energy cost of data movement.
(2) Most QNN operations are on few-bit integers, which are faster and more energy-efficient than floating-point.

While a quantized network will generally have reduced accuracy compared to an equivalent DNN using floating point, recent research has demonstrated significant progress in closing this accuracy gap. Courbariaux and Hubara et al. [4] first demonstrated that Binarized Neural Networks (BNNs), a QNN variant with $\mathbf{W^1 A^1}$, could achieve competitive accuracy on smaller image recognition benchmarks like CIFAR-10 and SVHN. XNOR-Net [6] improved upon this technique by adding scaling factors to better approximate the full-precision operations. Noting that more challenging classification tasks such as ImageNet could benefit from higher-precision activations, DoReFa-Net [8] used multi-bit activations and weights to further improve accuracy. Recently, Cai et al. [1] proposed Half-wave Gaussian Quantization (HWGQ) to take advantage of the Gaussian-like distribution of batch-normalized activations, demonstrating $\mathbf{W^1 A^2}$ networks with less than 5% top-5 accuracy drop compared to floating point DNNs on the challenging ImageNet dataset, as summarized in Table 1.

Table 1: Accuracy of a state-of-the-art QNN [1].

Dataset	Network	Floating Point top-1 (top-5)	$\mathbf{W^1 A^2}$ HWGQ [1] top-1 (top-5)
ImageNet	AlexNet	58.5% (81.5%)	52.7% (76.3%)
ImageNet	GoogLeNet	71.4% (90.5%)	63.0% (84.9%)
ImageNet	VGG-like	69.8% (89.3%)	64.1% (85.6%)
CIFAR-10	VGG-like	93.2%	92.5%

Despite the attractive accuracy and computational properties, there is a challenge in reaping the benefits on QNNs on mobile devices with commodity processors. Namely, operations on few-bit integers are not natively supported in most instruction set architectures (ISAs), which is the problem we address in this work.

2 QNN INFERENCE ON MOBILE DEVICES

The dominating computation in QNN inference is multiplications between few-bit integer matrices. These multiplications can be carried out by casting all operands to 8-bit integers, which are natively supported by most ISAs today. Libraries such as Google's gemmlowp [3], which has been used to deploy DNNs on mobile devices, offer high-performance 8-bit matrix multiplications. However, using 8-bit operands to carry out few-bit integer arithmetic can be wasteful. For instance, using 8-bit operations to compute a $\mathbf{W^2 A^2}$ matrix product would insert six zero bits into each operand, thus unnecessarily increasing the memory footprint by 4×.

As an alternative, we propose a method that uses commonly-supported bitwise operations for efficient few-bit integer matrix multiplication. We will first describe how this is done for the $\mathbf{W^1 A^1}$ case, then generalize the method to $\mathbf{W^w A^a}$.

The $\mathbf{W^1 A^1}$ case. Binary matrix multiplication, which we refer to as BinaryGEMM, can be used for the case where each weight and activations can be represented using a single bit. Previous work [4–6] discussed how binary dot products can be implemented using bitwise XNOR followed by popcount (counting the number of set bits) operations. Most modern processors provide an instruction for popcount, which enables fast BinaryGEMM implementations even on mobile CPUs. Note that very high performance (in the trillion-operations per second range) on these operations can also be achieved with FPGAs [5, 7] and GPGPUs [4]. Although XNOR-popcount is only applicable for matrices with $\{-1, +1\}$ binary elements, it is possible to extend this idea to $\{0, 1\}$ binary elements by using bitwise AND instead of XNOR as shown in Algorithm 1.

The $\mathbf{W^w A^a}$ case. We now leverage BinaryGEMM as a building block for implementing few-bit integer matrix multiplication. We can rewrite the w-bit matrix W as a weighted sum $\sum_{i=0}^{w-1} 2^i \cdot W[i]$,

```
function BinaryGEMM(W, A, res, α)
  for r ← 0 . . . rows − 1 do
    for c ← 0 . . . cols − 1 do
      for d ← 0 . . . ⌈depth/wordsize⌉ − 1 do
        res[r][c] += α · Popcount(W(r, d) & A(c, d))
      end for
    end for
  end for
end function
```

Algorithm 1: $\mathbf{W^1A^1}$ GEMM using AND-popcount.

```
function BitSerialGEMM(W, A, res)
  for i ← 0 . . . w − 1 do
    for j ← 0 . . . a − 1 do
      sgnW ← (i == w − 1 ? −1 : 1)
      sgnA ← (j == a − 1 ? −1 : 1)
      BinaryGEMM(W[i], A[j], res, sgnW · sgnA · 2^(i+j))
    end for
  end for
end function
```

Algorithm 2: Signed $\mathbf{W^wA^a}$ GEMM using BinaryGEMM.

where $W[i]$ is the binary matrix formed by taking bit i of each element of W, also referred to as a *bit plane*. In this manner, the product of two few-bit integer matrices can be written as a weighted sum of the pairwise products of their bit planes, i.e. $W \cdot A = \sum_{i=0}^{w-1} \sum_{j=0}^{a-1} 2^{i+j} \cdot W[i] \cdot A[j]$.

Algorithm 2 uses this observation to formulate few-bit integer matrix multiplication between the w-bit weight matrix W and the a-bit activation matrix A. This is a *vectorized bit-serial* operation, since the contributions to each result element are computed between two bit positions at a time, but the bitwise operations inside BinaryGEMM operate on vectors of bits. In this manner, we are able to take advantage of the full width of the processor datapath without introducing a large number of zero bits inside operations regardless of the values of w and a. The time taken by BitSerialGEMM will be proportional to $w \cdot a$, with $\mathbf{W^1A^1}$ executing fastest.

3 EVALUATION

We implemented BitSerialGEMM using ARM NEON intrinsics in C++, with register blocking and L1 cache blocking to achieve higher performance. We compare against the gemmlowp library [3], which utilizes hand-optimized inline assembly for 8-bit matrix multiplications. All reported results are from an LG Nexus 5X mobile phone running Android 7.1.2, using a single thread running on an ARM Cortex-A57 core at 1.8 GHz. For a matrix multiplication with dimensions (rows, depth, cols) that takes T nanoseconds, we report the performance in integer giga-operations per second (GOPS) measurement as $(2 \cdot \text{rows} \cdot \text{depth} \cdot \text{cols}) \,/\, T$ by averaging over a runtime of 10 s.

3.1 Compute-Bound Performance

We start by testing the compute-bound performance using matrices that fit into the L1 cache. For gemmlowp, we observed a peak performance of 22 GOPS. For BitSerialGEMM on $\mathbf{W^1A^1}$ (binary matrices), we observed a peak performance of 150 GOPS, which is 6.8× faster than using 8-bit operands. As expected, the performance linearly decreases with more bits of precision: 77 GOPS for $\mathbf{W^1A^2}$, 50 GOPS for $\mathbf{W^1A^3}$, 34 GOPS for $\mathbf{W^2A^2}$ and 23 GOPS for $\mathbf{W^2A^3}$.

Table 2: Key metrics on $\mathbf{W^1A^2}$ AlexNet with batch size 1.

Dimensions	Performance (GOPS)		L1D miss rate	
(rows, depth, cols)	8-bit	bit-serial	8-bit	bit-serial
(96, 363, 3025)	14.8	26.3	2.40%	1.66%
(256, 2400, 729)	14.1	41.8	5.18%	2.70%
(384, 2304, 169)	13.8	54.0	11.85%	5.02%
(384, 3456, 169)	14.1	58.3	11.63%	4.67%
(256, 3456, 169)	13.6	58.5	9.54%	4.68%
(4096, 9216, 1)	1.4	18.5	18.56%	9.40%
(4096, 4096, 1)	1.2	16.4	16.06%	11.70%
(1000, 4096, 1)	1.2	31.9	16.41%	15.33%
AlexNet GOPS	13.4	46.3		

Thus, for this particular platform, BitSerialGEMM is faster than using 8-bit operations for $\mathbf{W^wA^a}$ with $w \cdot a \leq 6.8$.

3.2 AlexNet matrices

We use a set of matrix multiplications from the $\mathbf{W^1A^2}$ AlexNet (1.13 GOP per frame) in [1] to compare BitSerialGEMM and gemmlowp on a realistic inference workload. Table 2 lists the per-layer and overall performance in GOPS, as well as the L1 data cache miss rate. The first five rows are convolutional layers with high arithmetic intensity, while the last three rows are fully-connected layers with low arithmetic intensity. On convolutional layers, BitSerialGEMM is up to 4× faster, with about half the cache misses of using 8-bit operations. The benefit due to smaller model size is especially prominent on the fully-connected layers with many parameters but few operations, where BitSerialGEMM is up to 26.6× faster than gemmlowp. Based on the overall performance and assuming 10% overhead due to other operations in the network, we project a $\mathbf{W^1A^2}$ AlexNet inference frame rate of 36.9 FPS for BitSerialGEMM and 10.7 FPS for gemmlowp. This performance can be further improved by multi-core parallelism and code optimization.

4 CONCLUSION AND FUTURE WORK

We have presented a method for performing few-bit integer multiplications using bitwise operations on commodity processors. Our preliminary results indicate that this method is faster and more efficient than using 8-bit operations for few-bit QNN inference. We note that this approach can enable approximate computing by only considering the contributions from higher-order bits and taking advantage of bit-level sparsity. Another use for this technique would be training DNNs using low-precision gradients [2], which also requires low-precision matrix operations. Future work will include more detailed performance characterization on end-to-end QNN inference implementations.

REFERENCES

[1] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. 2017. Deep Learning with Low Precision by Half-wave Gaussian Quantization. In *CVPR*.

[2] Christopher De Sa, Michael Feldman, Kunle Olukotun, and Christopher Re. 2017. Understanding and Optimizing Asynchronous Low-Precision Stochastic Gradient Descent. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE.

[3] Benoit Jacob et al. 2017. gemmlowp: a small self-contained low-precision GEMM library. (2017). https://github.com/google/gemmlowp.

[4] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized Neural Networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain.*

[5] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. 2016. Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC. In *Field-Programmable Technology (FPT), 2016 International Conference on*. IEEE.

[6] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *ECCV*.

[7] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17)*. ACM.

[8] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. 2016. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *CoRR* abs/1606.06160 (2016).