MPRI-JNSA | INTERNATIONAL JOURNAL OF NATURE AND SCIENCE ADVANCE RES.
VOL. 1 NO. 1 JUN-2022

A RESIDUE NUMBER SYSTEM BASED CONVOLUTION NEURAL NETWORK ALGORITHM

ABDULRAHMAN SHOLA AKANBI; DAMILOLA DAVID POPOOLA; & PROFESSOR KAZEEM ALAGBE GBOLAGADE

Department of Computer Science, Faculty of Information and Communication Technology, Kwara State University, Malete

Abstract

A number of positively convergent elements have aided the development of Deep Learning. The efficiency of floating point operations is highly optimized in modern micro-architectures. A whole area of research has emerged around quantized models, which reduce by orders of magnitude the amount of required memory, with a particular focus on quantized convolution neural networks. However, there is still a need to rethink how these quantized models can then be accelerated efficiently. The research starts by recognizing that inference in convolution neural networks is fundamentally a Digital Signal Processing (DSP) task. Memory, computation, and power utilization were expensive, in a different but similar way to how they are on modern mobile platforms, whose budget is set by their battery capacity. It is therefore of utmost importance to provide an alternative solution to this problem. This research introduces Residue Number System Architecture to the process to take advantage of the limited - but not binary - range of values that the operands can assume during the convolution operation in a quantized convolution neural network, to perform convolution more efficiently. This has reduced the memory footprint of neural networks, such that many off-chip memory accesses which dominate energy consumption during evaluation networks which is memory and computational intensive has also been reduce by parallel computation.

KEYWORDS: *Deep Learning, Neural Network, Convolution Neural Network, Digital Signal Processing,*

EDITOR-IN-CHIEF
Prof. E. McLenna [Austrilia]

INTRODUCTION

In the past, the development of Deep Learning has been sustained by a series of positively converging factors. Modern micro-architectures are highly optimized towards efficient floating point operations. General purpose parallel architectures nowadays are ubiquitous, thanks to the foresight of vendors such as Nvidia. Basic Linear Algebra Subprograms (BLAS), which powers many deep learning computations, are highly efficient, as a result of work of pioneering researchers such as Kazushige (Goto & Geijn, 2008). This has led the Deep Learning community to focus more on how to scale deeper models on increasingly parallel systems, while the underlying hardware and software building blocks have been treated as black boxes. More recently, the need to deploy Deep Learning models on power and resource constrained platforms have pushed researchers into looking for ways to build models that perform more efficiently considering the inference time, memory and computation. A whole area of research has emerged around quantized models, which reduce by orders of magnitude the amount of required memory, with a particular focus on quantized convolution neural networks. However, there is still a need to rethink how these quantized models can be accelerated. It start by recognizing that inference in convolution neural networks is fundamentally a Digital Signal Processing (DSP) task.

Convolution, which is at the heart of convolution neural network(CNN), is also at the heart of DSP. The study recognize that the constrains posed by mobile and embedded platforms on Deep Learning tasks have a lot in common with the constrains posed on DSP tasks during the second half of the last century by what were at the time state-of-the-art architectures. Memory, computation, and power utilization were expensive, in a different but yet similar way to how they are on modern mobile platforms, whose budget is set by their battery capacity. This led to a spur of research into fast algorithms (Cooley and Tukey 1965; Strassen, 1969; Karabutsa and Ofman 1962; Cook and Aanderaa, 1969; Winograd, 1976; Pollard, 1971; Schonhage and Strassen, 1971; Nussbaumer, 1980) are among the most relevant), which are traditionally defined as requiring a reduced number of additions and multiplications with respect to their naive counterpart.

In the area of quantized convolution neural network, a lot of focus has been given to models where both the activations and the weights are quantized to a binary value (Hubara *et al*, 2016), even though they tend to perform poorly compared to models quantized to bigger bit lengths. The reason is that completely binarize models shows not only drastic memory savings, but also algorithmic advantages,

EDITOR-IN-CHIEF
Prof. E. McLenna [Austrilia]

since convolution between binary sequences can be computed through a series of bit-wise operations (Rastegari *et al*, 2016). Once you move away from a completely binary representation, scaling these bit-wise kernels have to be applied multiple times and requires additional steps (Hubara et. al., 2016). The focus in reaching to the DSP literature has been then to try to provide an alternative solution to this problem, i.e. to take advantage of the limited - but not binary - range of values that the operands can assume during the convolution operation in a quantized convolution neural network, in order to perform convolution more efficiently.

Residue Number System (RNS) is an integer number system with the competency to support parallel, carry-free addition, borrow-free subtraction and single step multiplication without unfinished product. Data Conversion in RNS is usually based on either the Chinese Remainder Theorem (CRT)] or the Mixed Radix Conversion (MRC), which can be categorized into forward and backward conversion (Popoola, 2019).

Convolutional Neural Networks are well established Deep Learning models that reach state-of-the-art accuracy in many pattern recognition tasks. More recently, much research has been invested in trying to reduce the memory requirements of convents at inference time, by quantizing the convolution operands to a reduced amount of bits. In the case of completely binarize convolution this can also lead to significant computational savings, given that convolution between binary sequences requires only bit-wise operations. Otherwise, it is still possible to perform convolution through bit-wise operations, but the same kernel has to be applied over and over. For this reason it is desirable to present Residue Number System based Convolution Neural Network in order to reduce the computational time of CNN.

Literature Review

Different approaches have been taken in the literature. In general, the most important parameters are the chosen bit sizes for the quantized activations and weights.

The simplest approach is to perform a linear mapping to an interval of positive integers [0;], with 0 representing the minimum value found within the operands,] the maximum, and n the chosen bit size. This is what is done in (Hubara et. al., 2016) and also what Google implements in its Tensor Flow library. A more detailed description of the implementation can be found here (https://www.tensorflow.org 2017). A different approach which can result in computational savings is to perform a logarithmic mapping, as it has been shown in (Miyashita *et al*, 2016) Non-linear approaches can be taken too, such as is it done

EDITOR-IN-CHIEF
Prof. E. McLenna [Austrilia]

in (Han *et al*, 2015). Another important point is how the rounding is performed during the quantization process: for example, employing stochastic rounding makes a difference when the quantization is performed at training time (Gupta, *et al*, 2015).

Furthermore, it is a matter of balancing the chosen bit sizes with the most convenient representation in terms of memory and computational savings with respect to the underlying hardware and software. For example, half precision floating-point arithmetic is a recent standard and it's not supported natively on all platforms, meaning that in practice it may be implemented inefficiently and thus erases all the supposedly gained benefits. A good review of the different kind of representation is provided in (Courbariaux *et al*, 2014).

For what concerns the application of fast algorithms to convnets, two areas of research developed in the last few years. One that is closely related to the one presented here consists in the application of FFT algorithms. Relevant works are (Mathieu *el al*., 2013) and (Lavin and Gray, 2016), which show the feasibility of FFT algorithms for the convolution with medium to large filters. The other major area of research involves the application of Winograd-style algorithms to the computation of short convolutions, which has been introduced in (Lavin and Gray, 2016) and represents the state of the art in terms of performances on small filters. Both areas of research involve non-quantized convnets.

Regarding convolution in quantized convnets, in (Zhou *el al*., 2016) and (Hubara *et. al*., 2016) the idea of extending bit-wise convolution kernels to non-binary quantized convolution is explored. Google is developing a matrix multiplication library called Gemmlowp (Gemmlowp, 2017) that is specifically focused on computing matrix multiplication on operands quantized to a maximum of 8 bits. However, currently it is used to implement naive convolution only. Finally, regarding the implementation of NTTs, from undertsaing knowledge Microsoft Research's one (Longa and Naehrig, 2016) represents the state of the art in terms of both algorithmic and software optimizations for a NTT on a small prime field, although its focus is on a long, one dimensional transform that is suitable for the New Hope (Alkim *et al*, 2016) post-quantum cryptography scheme.

In general, the easiest approaches to keep good levels of accuracy is to first train a model in full precision, and then quantizes it, and finally perform an additional retraining. This is what is done in (Han *et al*, 2015). More recently, researchers have shown the feasibility of directly training a quantized model (Hubara *et al*., 2016).

EDITOR-IN-CHIEF
Prof. E. McLenna [Austrilia]

In this case, another important parameter to account for is the bit size to which the gradients are quantized.

Materials and Method

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs use a variation of multilayer perceptron designed to require minimal preprocessing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, RELU layer i.e. activation function, pooling layers, fully connected layers and normalization layers.

Convolutional Neural Network Based RNS

A Residue Number System (RNS) is a technique based on the Chinese Remainder Theorem (CRT) that allows representing a larger integer using a set of smaller integer. In particular, a RNS allows to split a computational on large integers into a set of parallel computations on smaller integers, that can be then combined together to reach the final result.

In the context of computing convolution using Number Theoretic Transforms (NTTs), RNS are especially useful to extend the maximum allowed output bit size by combining parallel NTTs. Given a set of NTTs modulo, using a RNS allows to extend the maximum represent able output value to . Modular arithmetic is performed with respect to each field's modulo, and the final combination step

EDITOR-IN-CHIEF
Prof. E. McLenna [Austrilia]

does not require modular arithmetic. The methodology can be applied to combine any NTTs with the same length.
NTTs never found widespread application in the DSP community. However, being exact fast algorithms, they were largely adopted by the Computational Number Theory community and, more recently, by the Cryptography community.

Algorithms

- *Step1: 2D Fermat Number Transform modulo F3 = 257.*
- *Step2: 2D Fermat Number Transform modulo F4 = 65537.*
- *Step3: Combination of 2D FNTs modulo F3 and F4 through a Residue Number System.*
- *Step4: 2D Overlap-and-save convolution based on blocks of circular convolution of size 8x8, 16x16, and 32x32.*

The maximum bit size for input ($b_{h;max}$) and kernel ($b_{k;max}$) depending on the chosen transform, kernel and block sizes. The block sizes refer to a convolution computed using an overlap-and-save 2D algorithm with no input padding (in a convnets sense). Because F3 allows for a maximum 1D transformation of length of 256, the 2D case the block size cannot go over 16 X 16. Since employing a RNS requires the transformation lengths to have the same size, RNS is limited to a maximum block size of 16 X 16 as well.
All the forward and inverse transform for the block sizes listed in the table are multiplier-less, meaning that they can all be implemented using a power-of-2 primitive root of unity. The bit sizes $b_{h;max}$ and $b_{k;max}$ have been obtained simply by maximizing both input and kernel values at increasing bit sizes, and checking for the correctness of the result. It stops at 8 bits, which is considered the threshold of interest for quantized convnets. Likewise, stop at filters of size 11 X 11, although blocks of size 32 X 32 support kernels up to size 17 X 17.
Subroutines
At a subroutine level, This scheme employs the following algorithms and optimizations:

- *Adopting Fast Fermat modular reduction.*
- *Avoid input reordering by using a Decimation in Frequency (DIF) algorithm for the forward transform, and a Decimation in Time (DIT) for the inverse transform. This strategy is also employed in (Creutzbury and Labunets, 1991).*

EDITOR-IN-CHIEF
Prof. E. McLenna [Austrilia]

- *1D NTTs using a radix-2 algorithm, based on the pseudo-code given in (Arndt, 2010), but with the modifications described below. The scheme gives the pseudo-code for both DIT (algorithm 1) and DIF (algorithm 2) implementation.*
- *Avoid stride memory accesses during the computation of forward and inverse NTTs with pre-computed twiddle factors. According to (Arndt, 2010), traditional radix-2 FFT implementations are plagued by stride memory access because they switch the order of the two innermost loops, given that this allows computing the twiddle factors in the outer loop and not in the inner one. This makes sense for traditional DSP applications that work on long transforms, since it greatly reduce the number of sin and cos to computes. In this case, however, this work is on short transforms with a handful of pre-computed factors, so it makes more sense to restore the original order of the loops.*
- *2D NTTs using the row-column approach,*
- *Perform element-wise product on the transposed input and kernel matrices, and transpose back only the result of the convolution. This strategy avoids useless transpositions and is employed in (Creutzbury and Labunets, 1991) as well.*
- *Perform the normalization step of the inverse NTT only on the output points not discarded by overlap-and-add.*
- *Pre-compute the twiddle factors for the forward and inverse transforms for lengths 8 and 16 for NTTs modulo F3, and for lengths 8, 16, and 32 for NTTs modulo F4, i.e. when the length allows to use a power-of-2 primitive root of unity.*
- *Compute the forward transform of the kernel only once per overlap-and-save call. Given that blocks are of small size, this has little memory impact.*

Results and Discussions

This research implemented a statically typed system in the following way. First, native integer types are mapped to a custom name using a typedef. A typedef allows to create a single point in the code to switch the underlying type representation:

- uint16_t ! qconv_inner_uint16: this type is the private representation of an unsigned 16bit integer.
- uint32_t ! qconv_inner_uint32: this type is the private representation of an unsigned 32bit integer.

EDITOR-IN-CHIEF
Prof. E. McLenna [Austrilia]

Because a typedef does not introduce any compile time restriction, This scheme uses one field struct to wrap the inner representation. This trick does not introduce any run-time overhead, but it forces the compiler to raise a warning when the wrong type is struct is used in place of another:

- qconv_uint16 {qconv_inner_uint16 value;}: this type is a struct and represents the public representation of an unsigned 16bit integer.
- qconv_uint16_mod_f_3 {qconv_inner_uint16 value;}: this type is a struct and represents the public representation of a modulo F3 unsigned integer.
- qconv_uint32 {qconv_inner_uint32 value;}: this type is a struct and represents the public representation of an unsigned 16bit integer.
- qconv_uint32_mod_f_4 {qconv_inner_uint32 value;}: this type is a struct and represents the public representation of a modulo F4 unsigned integer.

Finally, because a a 16bit (32bit) value may be interpreted both as unsigned 16bit (32bit) integer and as a modulo F3 (F4) value, The scheme overlay the different structs that share the same private representation by means of an union:

qconv_uint16_mod {
qconv_uint16 uint16 ;
qconv_uint16_mod_f_3 mod_f_3;
}: this type is an union and can be interpreted either as an unsigned 16bit integer or as modulo F3 unsigned integer.

- qconv_uint32_mod {
  qconv_uint32 uint32 ;
  qconv_uint32_mod_f_4 mod_f_4;
  }: this type is an union and can be interpreted either as an unsigned 32bit integer or as modulo F4 unsigned integer.

Fast Ferma Algorithm

The methodology requires implementing Fermat arithmetic. It would desirable to do so in an efficient way.

Modular Addition and Subtraction

Modular addition and subtraction can be implemented by means of two signed adds and a comparison with 0.

Modular Multiplication and Shift

The methodology requires implementing unsigned modular multiplication and signed modular shift. Signed modular shifts are required by the fact that, for a primitive root of unity $w = 2$, performing the inverse transform requires multiplications with powers of 2 taken with a negative sign, which is equivalent to negating the shifted operand - which is assumed to be non-negative and then shifting it by the corresponding power of 2 taken with a non-negative sign, which corresponds to a /textitsigned left shift. Signed left shift are undefined behaviour in C (while signed right shift are implementation defined behaviour). Because of that, after negating the operand to shift, bring it to a positive range, and finally shift it by the power of 2 taken with a non-negative sign. In practice, this can be implemented by first subtracting the value to shift from the modulo value, and then shifting it.

However, for both F4 multiplications and shifts, there is need to pay a branch instruction because of a corner case: given that multiplying or shifting an unsigned 16bit value by 16bit may overflow a 32bit unsigned integer; has to be treated separately. For multiplications, this means that if both operands are equal to F4 1, then return 1, which is the result for the multiplication already reduced. For shifts, fortunately 16bit shifts only appears in inverse transforms.

Because of the series of transformation described above, a shift by 16bit in an inverse transform context means returning the value to be shifted as the already reduced result. Besides these corner cases, the result of a modular shifts or multiplications must be reduced. For Fermat modules, this can be performed using a fast routine.

Fast Fermat Reduction

This research implements fast reduction modulo a Fermat prime using the following sequence of operations: AND, SHIFT, SUBTRACT, CONDITIONAL ADD. The idea is the following, inspired by the method. Compute y = x mod p, with x a positive unsigned number and p a Fermat prime. Given q and r the quotient and the remainder of the division by p-1, that y = r-q mod p. Because p-1 is a power of two and x is a positive unsigned number, that r = x ^ (p - 1), while q = x » (p - 1). Finally, after performing the subtraction y = r - q, the result is not completely reduced, so there is need to add p to any y < 0.

This sequence of operations can also be easily performed with vector instructions, supporting the idea that a platform specific parallel implementation should be explored.

EDITOR-IN-CHIEF
Prof. E. McLenna [Austrilia]

Benchmarks

This research compares the portable reference overlap-and-save NTT 2D convnets convolution implementation against a reference naive 2D convnets convolution implementation. The naive implementation is provided in QConv as well. The embedded test platform of choice is a Rasperry Pi Zero loaded with Raspbian Jessie Lite 2017-03-02. The compiler is GCC version 4.9.2, with O3 optimizations enabled.

This research considers F3 blocks of sizes 8×8 and 16×16, and F4 blocks of sizes 8×8, 16×16 and 32×32. The kernel sizes considered are 3×3, 5×5, 7×7, 9×9, 11×11 with depth = 1, while the input sizes considered are 60×60, 120×120, 224×224 with depth = 1. 224×224 is the input size of the first layer for many popular models, so it was chosen as the biggest input size to be tested. For each case, the bit sizes are maximized according to the bounds given in table Table I. This does not benchmark the RNS given that it's inherently unsuitable for a serial implementation and expect it to perform poorly. The benchmarks are an average over 20 runs. The results are presented.

Table 1: MAXIMUM INPUT AND KERNEL BIT SIZES

FNT	KERNEL	Min BLOCK	Max BLOCK	$b_{h,\,max}$	$b_{k,max}$
F3	3×3	8,8	16×16	3	2
F3	5×5	8×8	16×16	2	2
F3	7×7, 9×9	16×16	16×16	2	1
F4	3×3	8×8	32×32	6	6
F4	5×5	8×8	32×32	6	5
F4	7×7	16×16	32×32	5	5
F4	9×9	16×16	32×32	5	4
F4	11×11	32×32	32×32	5	4
RNS (F3,F4)	3×3, 5×5	8×8	16×16	8	8
RNS (F3,F4)	7×7, 9×9	16×16	16×16	8	8

Similarly to their complex counterpart, NTT performs better than the naive algorithm with larger input, kernel and block sizes. In particular, overlap-and-save works better with bigger inputs because it can computed more blocks without zero padding th input patches to reach a full block size, a step that is required around the right-end borders of the input. With input = 60×60, a tie between the naive implementation and F4 with block 32×32 is reached at kernels sized 11×11, and

from that on F4 with block 32×32 outperforms the direct implementation. With input = 120×120. F4 with block 32×32 outperforms the direct implementation starting from kernels sized 9×9. With input = 224×224, a tie between the naive implementation and F4 with block 32×32 is reached at kernel 7×7, and from that on F4 with block 32×32 outperforms the direct implementation.

The observed performance makes RNS suitable for the first layer of a convert, given that in state-of-the-art models such as ResNet. It tends to be larger to capture more information. Moreover, in a mobile context, the first layer may have to cover an image coming directly from the camera, so being able to support large input sizes makes sense. However, given that the first layer is usually stride, at the current state of things it would mean requiring a larger down sampling after the convolution, which - although it's usually performed anyway - means an increased cost in memory. On the other hand, it's also true that RNSs coupled with overlap-and save may have a lower memory impact that direct convolution implemented through img2col and GEMM: for a kernel dimension of length K with stride = 1, the corresponding RNS overlap-and-save implementation with a block dimension of length B employs img2col with stride = B - K+1, meaning that the overlap between the image patches extracted from the input to the working memory is greatly reduced.

Table 2: Theoretical cost in terms of operations per output point

Kernel	NAÏVE INT OPS/OUTPUT		BLOCK	VALID INPUT	NTT MODULO OPS/OUTPUT			
	ADD	MUL			ADD	MUL	MUL BY COST	SHIFT
3×3	9	9	8×8	6×6	21.34	1.78	1	7.12
			16×16	14×14	20.89	1.31	1	7.84
			32×32	30×30	22.76	1.14	1	9.10
5×5	25	25	8×8	4×4	48	4	1	16
			16×16	12×12	28.44	1.78	1	10.66
			32×32	28×28	26.12	1.31	1	10.45
7×7	49	49	16×16	10×10	40.96	2.56	1	15.36
			32×32	26×26	30.29	1.51	1	12.12
9×9	81	81	16×16	8×8	64	4	1	24
			32×32	24×24	35.56	1.78	1	14.22
11×11	121	121	32×32	22×22	42.31	2.12	1	16.93

EDITOR-IN-CHIEF
Prof. E. McLenna [Austrilia]

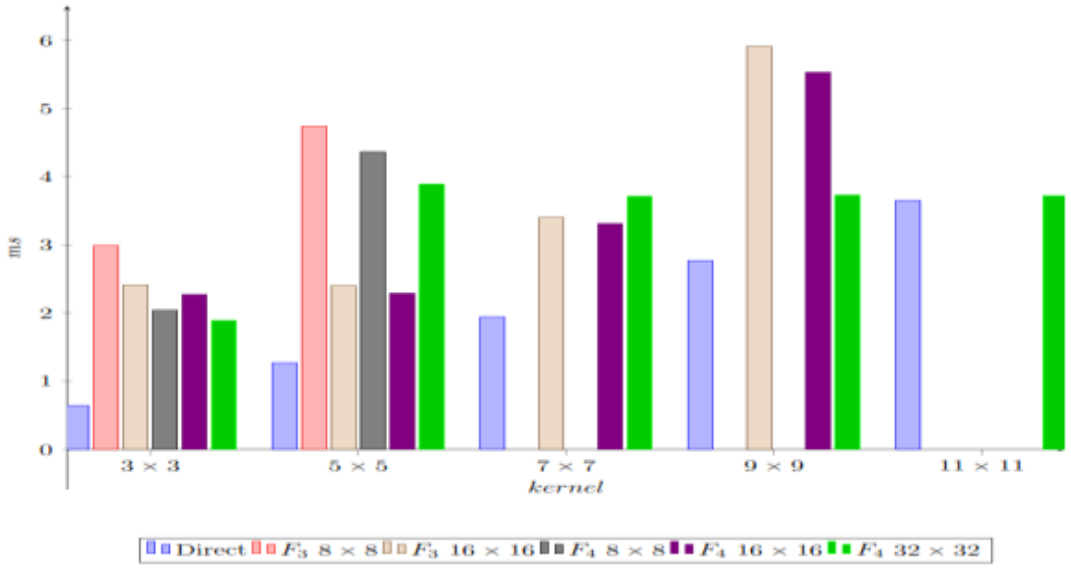

Figure 1: Overlap-and-save RNS convnets convolution on input = 60×60×1

Another suitable use case may be the 1x7 and 7x1 filters in the Inception-ResNet-v2 architecture which are not stride and being one dimensional carry little transformation overhead, the figure 4.1, 4.2 and 4.3 are Overlap-and-save RNS convnets convolution on input = 60×60×1, 120×120×1 and 224×224×1.

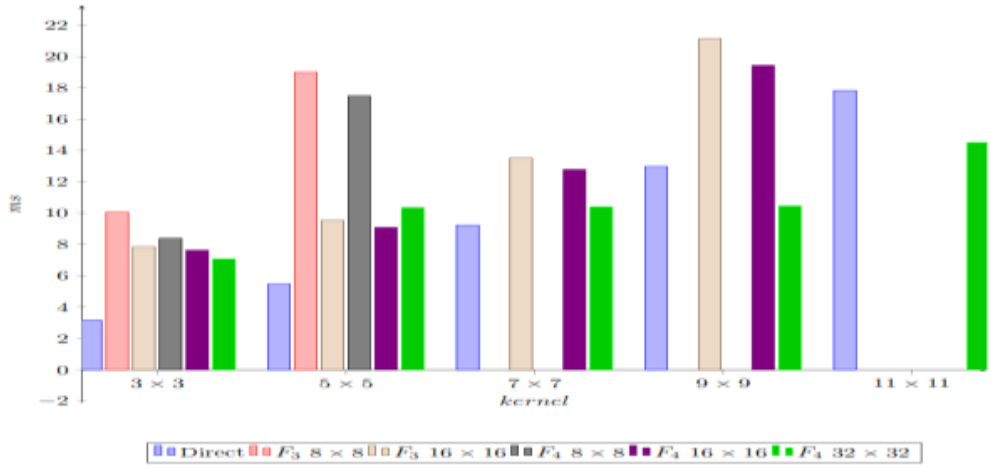

Figure 2: Overlap-and-save RNS convnets convolution on input = 120×120×1

EDITOR-IN-CHIEF
Prof. E. McLenna [Austrilia]

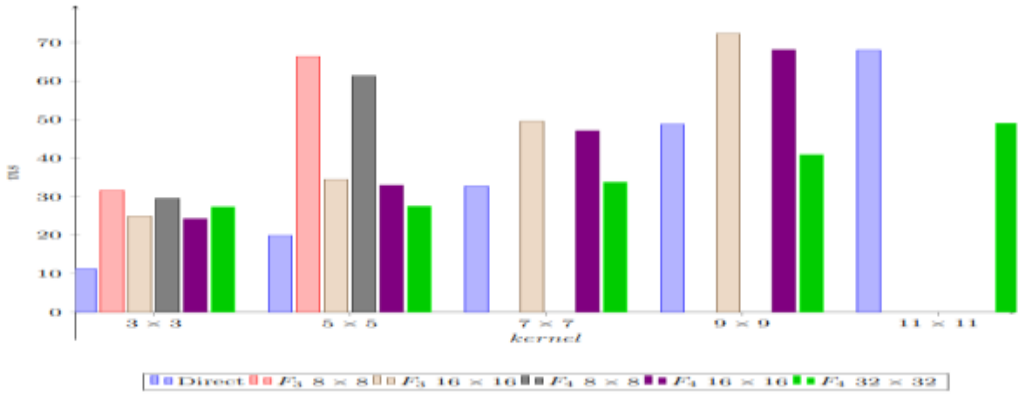

Figure 3: Overlap-and-save RNS convnets convolution on input = 224×224×1

Conclusion

This work introduced Residue Number System as a suitable fast algorithm to perform convolution in quantized convnets. it implemented two of such transforms, the 2D row-column Fermat Number Transforms modulo F3 and F4, as well as their combination through a Residue Number System. On top of the transform implementation, it implemented the 2D overlap-and-save algorithm to compute a block-by-block convolution. It gave theoretical costs of the computation. Also, listed a set of suitable bit sizes for the input and kernel depending on the convolution lengths. Finally, presented a list of cases in which the reference portable implementation performs better than a naive one, and a set of suitable use cases in which RNS may be employed. This a work in progress exploration of the topic.

Recommendation

A 2D New Mersenne Number Transform should be implemented and compared with the current FNTs. With respect to FFT algorithms, the following ones are still to be implemented and compared to the current implementation:

i. A 2D row-column FFT implementation using Winograd Fourier Transform Algorithm (WFTA) as the 1D subroutine.
ii. A 2D vector-radix FFT implementation.
iii. A 2D Nussbaumer-Quandalle FFT implementation.

In particular, with respect to FFTs, suitable transformations for stride convolution must be developed. From the software point of view, the following optimizations are still to be implemented:

EDITOR-IN-CHIEF
Prof. E. McLenna [Austrilia]

i. Lazy modular reduction for specific transformation lengths.
ii. Platform-specific SIMD implementation for a single FFT block.
iii. Parallel execution of multiple blocks in overlap-and-save.
iv. Parallel execution of different NTTs that are later combined using a RNS.

References

Alkim, E., Ducas, L., Pöppelmann, T., and Schwabe, P., (2016): Post-quantum key exchange-a new hope.

Cook, S. A. and Aanderaa, S. O., (1969): On the minimum computation time of functions. Transactions of the American Mathematical Society, 142:291_314.

Cooley, J. W. and Tukey, J. W., (1965): An algorithm for the machine calculation of complex fourier series. Mathematics of computation, 19(90):297_301, 1965.

Courbariaux, M., Bengio, Y., and David, J.-P., (2014): Training deep neural networks with low precision multiplications. arXiv preprint arXiv: 1412.7024.

Gemmlowp A., (2017) - low precision matrix multiplication. https://github.com/google/gemmlowp. Accessed.

Goto, K. and Geijn, R. A. (2008): Anatomy of high-performance matrix multiplication. ACM Transactions on Mathematical Software (TOMS), 34(3):12.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P., (2015): Deep learning with limited numerical precision. In ICML, pages 1737_1746.

Han, S., Mao, H., and Dally, W. J., (2015): Deep compression: Compressing deep neural networks with pruning, trained quantization and human coding. arXiv preprint arXiv: 1510.00149.

How to quantize neural networks with tensor flow. (2017) https://www.tensorflow.org/ performance/quantization. Accessed.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y., (2016): Quantized neural networks: Training neural networks with low precision weights and activations. arXiv preprint arXiv: 1609.07061.

Karabutsa, A. and Ofman, Y. (1962): Multiplication of many-digital numbers by automatic computers. DOKLADY AKADEMII NAUK SSSR, 145(2):293.

Lavin, A. and Gray, S., (2016): Fast algorithms for convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4013_ 4021.

Longa, P. and Naehrig, M., (2016): Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In International Conference on Cryptology and Network Security, pages 124_139. Springer.

Mathieu, M., Henaff, M., and LeCun, Y., (2013): Fast training of convolutional networks through _ts. arXiv preprint arXiv: 1312.5851.

Nussbaumer, H., (1980): Fast polynomial transform algorithms for digital convolution. IEEE Transactions on Acoustics, Speech, and Signal Processing, 28(2):205_215.

Pollard, J. M., (1971): The fast Fourier transform in a finite field. Mathematics of computation, 25(114):365_374.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A., (2016): Xnor-net: Imagenet classi_cation using binary convolutional neural networks. In European Conference on Computer Vision, pages 525_542. Springer.

EDITOR-IN-CHIEF
Prof. E. McLenna [Austrilia]

Schonhage, A. and Strassen, V., (1971): Fast multiplication of large numbers. Computing, 7(3- 4):281.
Strassen, V., (1969): Gaussian elimination is not optimal. Numerische mathematik, 13(4):354_356.
Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y., (2016): Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv: 1606.06160.

EDITOR-IN-CHIEF
Prof. E. McLenna [Austrilia]