Efficient Residue Number System Based Winograd Convolution

Zhi-Gang Liu and Matthew Mattina

Arm ML Research Lab, Boston, MA, USA
{Zhi-Gang.Liu,Matthew.Mattina}@arm.com

Abstract. Prior research has shown that Winograd algorithm can reduce the computational complexity of convolutional neural networks (CNN) with weights and activations represented in floating point. However it is difficult to apply the scheme to the inference of low-precision quantized (e.g. INT8) networks. Our work extends the Winograd algorithm to Residue Number System (RNS). The minimal complexity convolution is computed precisely over large transformation tile (e.g. 10×10 to 16×16) of filters and activation patches using the Winograd transformation and low cost (e.g. 8-bit) arithmetic without degrading the prediction accuracy of the networks during inference. The arithmetic complexity reduction is up to 7.03× while the performance improvement is up to 2.30× to 4.69× for 3×3 and 5×5 filters respectively.

1 Introduction

Machine learning has achieved great success in the past decade on a variety of applications including computer vision, natural language processing, and automatic speech recognition. In particular, deep convolutional neural networks (CNNs) have achieved better than human-level accuracy on image classification. The learning capability of CNNs improves with increasing depth and number of channels in the network layers. However this improvement comes at the expense of growing computation cost, particularly the expensive matrix or tensor multiplication and convolution. Thus reducing the computational complexity, especially the cost of the convolution operations, is critical for the deployment of these models on mobile and embedded devices with limited processing power.

Most recent CNN architectures [6] for image classification use low dimensional filters, typically 3×3, 5×5 or 7×7. The conventional Fast Fourier Transform (FFT) based convolution in the complex domain is inefficient with small filter dimensions. Faster algorithms for CNN inference based on Winograd minimal filters [14] can speed up the convolution by a factor of 2 to 4. The downside of the Winograd approach is that numerical problems and accuracy loss can occur unless high precision floating-point values are used.

Research on the quantization of neural network [2] [9] has shown that using reduced-precision representation (e.g. INT8) for the storage and computation of CNNs has significant benefits such as decreased memory bandwidth, lower memory foot-print, lower power consumption and higher throughput, while only

having a negligible prediction accuracy degradation. The predominant numerical format used for training neural networks is IEEE floating-point format (FP32). There is a potential 4× reduction in memory bandwidth and storage achieved by quantizing FP32 weights and activations to INT8 values. The corresponding energy and area saving are 13.5× and 27.3× [3] respectively. But, both Winograd and FFT methods [7] [10] require high precision arithmetic to avoid prediction accuracy degradation and are therefore non-ideal for improving low-precision integer e.g. INT8 convolution efficiently.

In this paper, we extend the Winograd minimal convolution [14] to Residue Number System (RNS) [12] targeting the inference of low-precision e.g. INT8 quantized convolutional neural networks. The key contributions are summarized here:

- We formulate the Winograd minimal complexity integer convolution over Residue Number System (RNS). The use of the RNS enables our algorithm to operate on quantized, low-precision e.g. INT8 CNNs with low cost, low precision integer arithmetic, without computational instability issues and without impacting the accuracy of the networks.
- Our RNS-based formulation enables the use of much larger Winograd transformation tiles, e.g. from 8x8 to 16x16. The theoretical arithmetic reduction is up to 2.3× and 4.69× for 3×3 and 5×5 filters respectively over 3-residue power efficient 8-bit RNS; 3.45× and 7.03× for 2-residue 16-bit RNS.
- We analyzed the performance with 8-bit quantized VGG16 models and show 2.02× to 2.2× improvement of inference latency on Arm Cortex-A73 CPU.

2 Related Work

Earlier work applied the classical FFT to speedup convolutional layers by reducing the arithmetic complexity [10]. This approach requires float arithmetic in the complex number $\mathbf{C}$, and multiplication involves the real and imaginary parts of complex value. A product of two complex values needs 3 or 4 floating multiplications, which is inefficient, especially for the small size filters commonly defined in popular CNNs.

The Winograd minimal filtering algorithm [14], first applied to CNNs by Lavin and Gray [7], can reduce arithmetic complexity from 2.25× to 4× for typical 3×3 CNN filters. However, the algorithm requires high precision arithmetic and hits computational instability issues when applied to large transform tile sizes [1]. An efficient sparse implementation of Winograd convolution have also been proposed [8]. The conventional Winograd convolution algorithm, including the latest enhancements, requires high precision floating point arithmetic.

Meanwhile, some researchers have tried to extend the Winograd algorithm to reduced-precision integer arithmetic by choosing complex interpolation points [11] with a 17.37% throughput improvement claimed, however it depends on a lossy precision scaling scheme, which would cause predication accuracy drop.

3 Residue Number System (RNS)

A Residue Number System, RNS($m_0, m_1, .., m_{n-1}$) [12], is number system to represent an integer by its value modulo n pairwise coprime moduli $m_0, m_1, ..,$ and m_{n-1}.

$$\begin{aligned} x_0 &= x \pmod{m_0} \\ x_1 &= x \pmod{m_1} \\ &... \\ x_{n-1} &= x \pmod{m_{n-1}} \end{aligned}$$

For example, to represent $x = 48$ in RNS($m_0 = 7, m_1 = 9$)

$$\{x \pmod{m_0}, x \pmod{m_1}\} = \{6, 3\}$$

We can construct the value of x from its RNS representation as long as $x < M$, where $M = \prod_{i=1}^{n-1} m_i$ is the dynamic range of the RNS($m_0, m_1, .., m_{n-1}$). For example, to convert $\{6, 3\}$ from RNS(7,9) back to standard form using Mixed Radix Conversion(MRC) [13] or Chinese Remainder Theorem (CRT) [5].

$$x = \left[6 + 7 * [\frac{3-6}{7} \pmod 9)]\right] \pmod{7 * 9} = 6 + 7 * 6 = 48$$

For addition($+$), subtraction($-$) and multiplication($*$) of two RNS values $x = \{x_0, x_1, .., x_{n-1}\}$ and $y = \{y_0, y_1, .., y_{n-1}\}$, it's sufficient to perform the operation on corresponding pair of residues. For example, $x = \{6, 3\}, y = \{5, 10\} \in$ RNS(7,9)

$$\begin{aligned} x + y &= \{6 + 5 \pmod 7, 3 + 10 \pmod 9\} = \{4, 4\} \\ x - y &= \{6 - 5 \pmod 7, 3 - 10 \pmod 9\} = \{1, 2\} \\ x * y &= \{6 * 5 \pmod 7, 3 * 10 \pmod 9\} = \{2, 3\} \end{aligned}$$

3.1 Convolution in RNS

Equivalently, we could calculate the convolution y of N-element vector $\boldsymbol{d} = (d_0, d_1, d_2, .., d_{N-1})$ and R-element filter $\boldsymbol{g} = (g_0, g_1, g_2, .., g_{R-1})$ over RNS(m_0, m_1, .., m_{n-1}).

$$\boldsymbol{y} = (y_0, y_1, y_2, .., y_{N-R}) = \boldsymbol{d} \circledast \boldsymbol{g}$$

and $y_k = \{y_k^{(0)}, y_k^{(1)}, .., y_k^{(n-1)}\} \in$ RNS($m_0, m_1, .., m_{n-1}$), where

$$y_k^{(i)} = (\sum_{j=0}^{R-1} d_{k+j} * g_j) \pmod{m_i}$$

4 Winograd Convolution

The Winograd convolution [14] is an optimal algorithm to compute short convolution over real numbers, outperforming conventional Discrete Fourier Transform (DFT). $F(M,R)$ denotes the convolution computation of M-tuple output y of a R-tuple filter g and N-tuple input d where $N = M + R - 1$. The Winograd algorithm calculates the $F(M,R)$ in a bilinear form as

$$y = A^T\Big[(Gg) \odot (B^T d)\Big]$$

where $\odot$ acts as element-wise production and B^T, G and A^T are $N \times N$, $N \times R$ and $M \times N$ transform matrices respectively.

Specifically, A^T, G and B^T are derived from the Vandermonde matrix [1] V generated from N distinct Lagrange interpolation points $S_0, S_1, S_2, ..S_{N-1}$ (Note: Require a special handling if $S_{N-1} = \infty$).

$$V = \begin{pmatrix} 1 & S_0 & S_0^2 & ... & S_0^{N-1} \\ 1 & S_1 & S_1^2 & ... & S_1^{N-1} \\ 1 & S_2 & S_2^2 & ... & S_2^{N-1} \\ .. & .. & .. & ... & .. \\ 1 & S_{N-1} & S_{N-1}^2 & ... & S_{N-1}^{N-1} \end{pmatrix}_{N \times N} \tag{1}$$

and

$$\begin{aligned} A^T &= V^T{}_{[0:M-1;0:N-1]} \\ G &= V_{\ [0:N-1;0:R-1]} \\ B^T &= V^{-T} \end{aligned}$$

For 2-D convolution, similar fast algorithm $F(M \times M, R \times R)$ can be represented as

$$y = A^T\Big[(GgG^T) \odot (B^T dB)\Big]A \tag{2}$$

We call GgG^T and $B^T dB$ the *forward transform* and $A^T[\cdot]A$ the *backward transform*.

Assuming the computation cost of transformation GgG^T and $B^T dB$ was amortized completely due to reuse, the fast algorithm requires $N^2 = (M + R - 1)^2$ multiplications while the standard method uses M^2R^2. The arithmetic complexity reduction is $\frac{M^2R^2}{(M+R-1)^2}$. For example:

$F(2 \times 2, 3 \times 3)$ with interpolation points $\{0, \pm 1, \infty\}$. The fractions in B^T are arranged into matrix G. The arithmetic complexity reduction is $2.25\times$.

$$A^T = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 1 \end{pmatrix};\quad B^T = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix};\quad G = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{-1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{pmatrix} = \frac{1}{2}G';\quad G' = \begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & 2 \end{pmatrix}$$

[1] https://en.wikipedia.org/wiki/Vandermonde_matrix

Table 1. The required data width of transformation and the corresponding arithmetic reduction for integer (INT8) Wingograd convolution algorithms. *DW* is transformation data width in bit. *Arithmetic Reduction* is the reduction of operation in *DW* bits.

Algorithm	DW (bit)	Arithmetic Reduction
$F(2\times 2,\ 3\times 3)$	12	2.25×
$F(4\times 4,\ 3\times 3)$	18	4.00×
$F(6\times 6,\ 3\times 3)$	24	5.06×
$F(8\times 8,\ 3\times 3)$	36	5.76×
$F(8\times 8,\ 5\times 5)$	43	11.1×
$F(10\times 10,\ 3\times 3)$	50	6.26×
$F(10\times 10,\ 5\times 5)$	60	12.7×

$F(4\times 4, 3\times 3)$ with interpolation points $\{0, \pm 1, \pm 2, \infty\}$. The arithmetic complexity reduction is $4\times$.

$$A^T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 2 & -2 & 0 \\ 0 & 1 & 1 & 4 & 4 & 0 \\ 0 & 1 & -1 & 8 & -8 & 1 \end{pmatrix};\ B^T = \begin{pmatrix} 4 & 0 & -5 & 0 & 1 & 0 \\ 0 & 4 & 4 & -1 & -1 & 0 \\ 0 & -4 & 4 & 1 & -1 & 0 \\ 0 & -2 & -1 & 2 & 1 & 0 \\ 0 & 2 & -1 & -2 & 1 & 0 \\ 0 & 4 & 0 & -5 & 0 & 1 \end{pmatrix};\ G = \begin{pmatrix} \frac{1}{4} & 0 & 0 \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{-1}{6} & \frac{1}{6} \\ \frac{1}{24} & \frac{1}{12} & \frac{1}{6} \\ \frac{1}{24} & \frac{-1}{12} & \frac{1}{6} \\ 0 & 0 & 1 \end{pmatrix} = \frac{1}{24} G';\ G' = \begin{pmatrix} 6 & 0 & 0 \\ 4 & 4 & 4 \\ 4 & -4 & 4 \\ 1 & 2 & 4 \\ 1 & -2 & 4 \\ 0 & 0 & 24 \end{pmatrix}$$

where matrices A^T, G and B^T are derived from Vandermonde matrix of the roots to construct the transform.

$F(2\times 2, 3\times 3)$ and $F(4\times 4, 3\times 3)$ have theoretical arithmetic complexity reduction of $2.25\times$ and $4\times$ respectively. We can achieve the expected speedup using floating-point operation i.e. FP32. However, it's a challenge to implement the Winograd convolution using low-precision integral arithmetic for quantized CNN. To calculate exact convolution using integer arithmetic, we can obtain matrix G' by factoring out the common fraction α, e.g. $\alpha = \frac{1}{2}$ for $F(2\times 2, 3\times 3)$ and $\alpha = \frac{1}{24}$ for $F(4\times 4, 3\times 3)$, from corresponding matrix G. Then eq. 2 becomes $y = \alpha^2 A^T \Big[(G' g {G'}^T) \odot (B^T d B) \Big]$.

The magnitude of element in transformation $G' g {G'}^T$ and $B^T d B$ would be $\frac{trace(G' {G'}^T)}{N}$ and $\frac{trace(B^T B)}{N}$ times as large as the quantity of filter g and input d on average. Particularly, the magnification are $3.5\times$ and $2\times$ for $F(2\times 2, 3\times 3)$ and $125\times$ and $28.7\times$ for $F(4\times 4, 3\times 3)$. Moreover, the magnifications we calculated correspond to the standard deviation statistically, the outliers could have much larger magnitudes. Practically, we need 12 bits to hold each element of transformation and INT16 arithmetic for element-wise multiply for $F(2\times 2, 3\times 3)$. $F(4\times 4, 4\times 3)$ demands 18 bits for transformation and INT32 arithmetic operations. We summarized the data width of transformation and arithmetic reduction of integer Winograd algorithms in table 1. Although the Winograd algorithms enable complexity reduction, they require higher precision arithmetic than INT8. Considering INT8 multipliers cost about $\frac{1}{4}$ power and area of INT16 case; $\frac{1}{15}$ and $\frac{1}{12}$ of INT32; $\frac{1}{18}$ and $\frac{1}{27}$ of FP32 respectively [3], there will be advantage in implementing the Winograd algorithm using INT8 arithmetic. For

this reason, a lossy precision scaling scheme was proposed [11], which scales down the transformation in the range of the desired low-precision arithmetic operation. However the scaling method introduces errors to the convolution output and would cause predication accuracy degradation.

The fundamental difficulty with performing the standard Winograd algorithm using low cost integral arithmetic is due to the ill-conditioned Vandermonde (and its inverse) matrix V in eq. 1 with real interpolation points especially for large transformation (e.g. M $>$ 6). We propose a different approach to implement the Winograd algorithm over Residue Number System (RNS) via low-precision integer arithmetic (e.g. INT8 or INT16) in the next section.

5 Winograd Convolution over Residue Number System

We extend the Winograd convolution algorithm described in section 4 to Residue Number System (RNS) in section 3 to formulate a new implementation. This new approach solves the numerical stability issue of the conventional Winograd algorithm for large transformation, i.e. $M \in [8, 16]$, moreover the new method is compatible with low precision 8-bit multiply and accumulation.

To simplify the description, without loss of generality, we take $F(10\times10, 3\times3)$ with interpolation points $\{0, \pm1, \pm2, \pm3, \pm4, \pm5, \infty\}$ as a running example with the following transform matrices A^T, B^T and G.

$$A^T = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & -1 & 2 & -2 & 3 & -3 & 4 & -4 & 5 & -5 & 0 \\
0 & 1 & 1 & 4 & 4 & 9 & 9 & 16 & 16 & 25 & 25 & 0 \\
0 & 1 & -1 & 8 & -8 & 27 & -27 & 64 & -64 & 125 & -125 & 0 \\
0 & 1 & 1 & 16 & 16 & 81 & 81 & 256 & 256 & 625 & 625 & 0 \\
0 & 1 & -1 & 32 & -32 & 243 & -243 & 1024 & -1024 & 3125 & -3125 & 0 \\
0 & 1 & 1 & 64 & 64 & 729 & 729 & 4096 & 4096 & 15625 & 15625 & 0 \\
0 & 1 & -1 & 128 & -128 & 2187 & -2187 & 16384 & -16384 & 78125 & -78125 & 0 \\
0 & 1 & 1 & 256 & 256 & 6561 & 6561 & 65536 & 65536 & 390625 & 390625 & 0 \\
0 & 1 & -1 & 512 & -512 & 19683 & -19683 & 262144 & -262144 & 1953125 & -1953125 & 1
\end{pmatrix}$$

$$B^T = \begin{pmatrix}
14400 & 0 & -21076 & 0 & 7645 & 0 & -1023 & 0 & 55 & 0 & -1 & 0 \\
0 & 14400 & 14400 & -6676 & -6676 & 969 & 969 & -54 & -54 & 1 & 1 & 0 \\
0 & -14400 & 14400 & 6676 & -6676 & -969 & 969 & 54 & -54 & -1 & 1 & 0 \\
0 & -7200 & -3600 & 8738 & 4369 & -1638 & -819 & 102 & 51 & -2 & -1 & 0 \\
0 & 7200 & -3600 & -8738 & 4369 & 1638 & -819 & -102 & 51 & 2 & -1 & 0 \\
0 & 4800 & 1600 & -6492 & -2164 & 1827 & 609 & -138 & -46 & 3 & 1 & 0 \\
0 & -4800 & 1600 & 6492 & -2164 & -1827 & 609 & 138 & -46 & -3 & 1 & 0 \\
0 & -3600 & -900 & 5044 & 1261 & -1596 & -399 & 156 & 39 & -4 & -1 & 0 \\
0 & 3600 & -900 & -5044 & 1261 & 1596 & -399 & -156 & 39 & 4 & -1 & 0 \\
0 & 2880 & 576 & -4100 & -820 & 1365 & 273 & -150 & -30 & 5 & 1 & 0 \\
0 & -2880 & 576 & 4100 & -820 & -1365 & 273 & 150 & -30 & -5 & 1 & 0 \\
0 & -14400 & 0 & 21076 & 0 & -7645 & 0 & 1023 & 0 & -55 & 0 & 1
\end{pmatrix}$$

$$G = \begin{pmatrix}
\frac{1}{14400} & 0 & 0 \\
\frac{1}{17280} & \frac{1}{17280} & \frac{1}{17280} \\
\frac{1}{17280} & \frac{-1}{17280} & \frac{1}{17280} \\
\frac{1}{30240} & \frac{1}{15120} & \frac{1}{7560} \\
\frac{1}{30240} & \frac{-1}{15120} & \frac{1}{7560} \\
\frac{1}{80640} & \frac{1}{26880} & \frac{1}{8960} \\
\frac{1}{80640} & \frac{-1}{26880} & \frac{1}{8960} \\
\frac{1}{362880} & \frac{1}{90720} & \frac{1}{22680} \\
\frac{1}{362880} & \frac{-1}{90720} & \frac{1}{22680} \\
\frac{1}{3628800} & \frac{1}{725760} & \frac{1}{145152} \\
\frac{1}{3628800} & \frac{-1}{725760} & \frac{1}{145152} \\
0 & 0 & 1
\end{pmatrix} = \frac{1}{3628800} G' ; \quad G' = \begin{pmatrix}
252 & 0 & 0 \\
210 & 210 & 210 \\
210 & -210 & 210 \\
120 & 240 & 480 \\
120 & -240 & 480 \\
45 & 135 & 405 \\
45 & -135 & 405 \\
10 & 40 & 160 \\
10 & -40 & 160 \\
1 & 5 & 25 \\
1 & -5 & 25 \\
0 & 0 & 3628800
\end{pmatrix}$$

These transforms are derived from the 12×12 Vandermonde matrix and its inverse matrix [2], which are not computationally friendly in standard number systems, including FP32 due to its numerical instability. However, we could mitigate the instability by carrying out the computation of eq. 2 over RNS$(m_0, m_1, .., m_{n-1})$.

To represent the transform matrix G in RNS, the modulus $m_0, m_1, ..,$ and m_{n-1} need be coprime to $\frac{1}{\alpha}$, e.g. $\frac{1}{\alpha} = 3628800 = 2^8 \cdot 3^4 \cdot 5^2 \cdot 7$ for the $F(10 \times 10, 3 \times 3)$ example.

Generically, the inverse of $N \times N$ Vandermonde matrix V in eq. 1 [2] [4], $V^{-1} = \{V_{i,j}^{-1}\}$, and $i, j \in [0, N-1]$ and $V_{i,j}^{-1}$ is given in eq. 3.

$$V_{i,j}^{-1} = \begin{cases} \frac{1}{\prod\limits_{m=0,\ m \neq j}^{N-1} (S_j - S_m)} & for\ j = N-1 \\ \frac{(-1)^{N-1-i} \sum\limits_{0 \leq j_0 < j_1 < \ldots < j_{N-1-i} < N,\ j_k \neq j} S_{j_0} S_{j_1} \ldots S_{j_{N-1-i}}}{\prod\limits_{m=0, m \neq j}^{N-1} (S_j - S_m)} & otherwise \end{cases} \tag{3}$$

where $S_0, S_1, S_2, ..., S_{N-1}$ are the interpolation points we choose to construct the Winograd transform. To obtain the multiplicative inverse of the denominator of $V_{i,j}^{-1}$ in eq. 3, each modulus m_i need be coprime to the denominator $\prod\limits_{m=0,\ m \neq j}^{N-1} (S_j - S_m)$.

For our example, the denominators in G are $14400 = 2^6 \cdot 3^2 \cdot 5^2$, $17280 = 2^7 \cdot 3^3 \cdot 5$, $30240 = 2^5 \cdot 3^5 \cdot 5 \cdot 7$, $80640 = 2^8 \cdot 3^2 \cdot 5 \cdot 7$, $362880 = 2^7 \cdot 3^4 \cdot 5 \cdot 7$ and $3628800 = 2^8 \cdot 3^4 \cdot 5^2 \cdot 7$. We chose moduli $m_0 = 11 \times 23 = 253$, $m_1 = 251$ and $m_2 = 13 \times 19 = 247$, which are all coprime to the denominators in G. Therefore the fractions in matrix G are all well-defined for modular division, for instance $\frac{1}{14400} \pmod{253} = 12$ as a result of multiplicative inverse of denominator, e.g. $14400 \times 12 \pmod{253} = 1$. Similarly, $\frac{1}{14400} \pmod{251} = 27$ and $\frac{1}{14400} \pmod{247} = -10$. Moreover, moduli $(253, 251, 247)$ are the largest suitable 8-bit values for the interpolation points we chose. Given that we can convert matrix A^T, G and B^T to corresponding modular format, e.g. $A_{m_i}^T = A^T \pmod{m_i}$, $G_{m_i} = G \pmod{m_i}$ and $B_{m_i}^T = B^T \pmod{m_i}$, where $m_i \in (253, 251, 247)$. The RNS representation of eq. 2 is

$$\begin{aligned} y = (&A_{253}^T \Big[[G_{253} g G_{253}^T] \odot [B_{253}^T d B_{253}] \Big] A_{253}, \\ &A_{251}^T \Big[[G_{251} g G_{251}^T] \odot [B_{251}^T d B_{251}] \Big] A_{251}, \\ &A_{247}^T \Big[[G_{247} g G_{247}^T] \odot [B_{247}^T d B_{247}] \Big] A_{247}) \end{aligned} \tag{4}$$

[2] https://proofwiki.org/wiki/Inverse_of_Vandermonde_Matrix

For modulo 253, the corresponding transform matrices are

$$A^T_{253} = \begin{pmatrix} 1&1&1&1&1&1&1&1&1&1&1&0\\ 0&1&-1&2&-2&3&-3&4&-4&5&-5&0\\ 0&1&1&4&4&9&9&16&16&25&25&0\\ 0&1&-1&8&-8&27&-27&64&-64&125&-125&0\\ 0&1&1&16&16&81&81&3&3&119&119&0\\ 0&1&-1&32&-32&-10&10&12&-12&89&-89&0\\ 0&1&1&64&64&-30&-30&48&48&-61&-61&0\\ 0&1&-1&-125&125&-90&90&-61&61&-52&52&0\\ 0&1&1&3&3&-17&-17&9&9&-7&-7&0\\ 0&1&-1&6&-6&-51&51&36&-36&-35&35&1 \end{pmatrix}; \quad G_{253} = \begin{pmatrix} 12&0&0\\ 10&10&10\\ 10&-10&10\\ 78&-97&59\\ 78&97&59\\ -34&-102&-53\\ -34&102&-53\\ -120&26&104\\ -120&-26&104\\ -12&-60&-47\\ -12&60&-47\\ 0&0&1 \end{pmatrix}$$

$$B^T_{253} = \begin{pmatrix} -21&0&-77&0&55&0&-11&0&55&0&-1&0\\ 0&-21&-21&-98&-98&-43&-43&-54&-54&1&1&0\\ 0&21&-21&98&-98&43&-43&54&-54&-1&1&0\\ 0&-116&-58&-117&68&-120&-60&102&51&-2&-1&0\\ 0&116&-58&117&68&120&-60&-102&51&2&-1&0\\ 0&-7&82&86&113&56&103&115&-46&3&1&0\\ 0&7&82&-86&113&-56&103&-115&-46&-3&1&0\\ 0&-58&112&-16&-4&-78&107&-97&39&-4&-1&0\\ 0&58&112&16&-4&78&107&97&39&4&-1&0\\ 0&97&70&-52&-61&100&20&103&-30&5&1&0\\ 0&-97&70&52&-61&-100&20&-103&-30&-5&1&0\\ 0&21&0&77&0&-55&0&11&0&-55&0&1 \end{pmatrix}$$

All elements in these matrices are in the range of $[-\frac{253-1}{2}, \frac{253-1}{2}]$. The computation of the fast convolution over (mod 253) can be performed with 8-bit low cost arithmetic operation without numerical concerns. Similarly, we can get the transforms for 251 and 247.

$$A^T_{251} = \begin{pmatrix} 1&1&1&1&1&1&1&1&1&1&1&0\\ 0&1&-1&2&-2&3&-3&4&-4&5&-5&0\\ 0&1&1&4&4&9&9&16&16&25&25&0\\ 0&1&-1&8&-8&27&-27&64&-64&125&-125&0\\ 0&1&1&16&16&81&81&5&5&123&123&0\\ 0&1&-1&32&-32&-8&8&20&-20&113&-113&0\\ 0&1&1&64&64&-24&-24&80&80&63&63&0\\ 0&1&-1&-123&123&-72&72&69&-69&64&-64&0\\ 0&1&1&5&5&35&35&25&25&69&69&0\\ 0&1&-1&10&-10&105&-105&100&-100&94&-94&1 \end{pmatrix}; \quad G_{251} = \begin{pmatrix} 27&0&0\\ -103&-103&-103\\ -103&103&-103\\ -23&-46&-92\\ -23&46&-92\\ -40&-120&-109\\ -40&120&-109\\ 19&76&53\\ 19&-76&53\\ 27&-116&-78\\ 27&116&-78\\ 0&0&1 \end{pmatrix}$$

$$B^T_{251} = \begin{pmatrix} 93&0&8&0&115&0&-19&0&55&0&-1&0\\ 0&93&93&101&101&-35&-35&-54&-54&1&1&0\\ 0&-93&93&-101&101&35&-35&54&-54&-1&1&0\\ 0&79&-86&-47&102&119&-66&102&51&-2&-1&0\\ 0&-79&-86&47&102&-119&-66&-102&51&2&-1&0\\ 0&31&94&34&95&70&107&113&-46&3&1&0\\ 0&-31&94&-34&95&-70&107&-113&-46&-3&1&0\\ 0&-86&104&24&6&-90&103&-95&39&-4&-1&0\\ 0&86&104&-24&6&90&103&95&39&4&-1&0\\ 0&119&74&-84&-67&110&22&101&-30&5&1&0\\ 0&-119&74&84&-67&-110&22&-101&-30&-5&1&0\\ 0&-93&0&-8&0&-115&0&19&0&-55&0&1 \end{pmatrix}$$

$$A^T_{247} = \begin{pmatrix} 1&1&1&1&1&1&1&1&1&1&1&0\\ 0&1&-1&2&-2&3&-3&4&-4&5&-5&0\\ 0&1&1&4&4&9&9&16&16&25&25&0\\ 0&1&-1&8&-8&27&-27&64&-64&-122&122&0\\ 0&1&1&16&16&81&81&9&9&-116&-116&0\\ 0&1&-1&32&-32&-4&4&36&-36&-86&86&0\\ 0&1&1&64&64&-12&-12&-103&-103&64&64&0\\ 0&1&-1&-119&119&-36&36&82&-82&73&-73&0\\ 0&1&1&9&9&-108&-108&81&81&118&118&0\\ 0&1&-1&18&-18&-77&77&77&-77&96&-96&1 \end{pmatrix}; \quad G_{247} = \begin{pmatrix} -10&0&0\\ 74&74&74\\ 74&-74&74\\ 7&14&28\\ 7&-14&28\\ -90&-23&-69\\ -90&23&-69\\ -20&-80&-73\\ -20&80&-73\\ -2&-10&-50\\ -2&10&-50\\ 0&0&1 \end{pmatrix}$$

$$B^T_{247} = \begin{pmatrix} 74&0&-81&0&-12&0&-35&0&55&0&-1&0\\ 0&74&74&-7&-7&-19&-19&-54&-54&1&1&0\\ 0&-74&74&7&-7&19&-19&54&-54&-1&1&0\\ 0&-37&105&93&-77&91&-78&102&51&-2&-1&0\\ 0&37&105&-93&-77&-91&-78&-102&51&2&-1&0\\ 0&107&118&-70&59&98&115&109&-46&3&1&0\\ 0&-107&118&70&59&-98&115&-109&-46&-3&1&0\\ 0&105&88&104&26&-114&95&-91&39&-4&-1&0\\ 0&-105&88&-104&26&114&95&91&39&4&-1&0\\ 0&-84&82&99&-79&-117&26&97&-30&5&1&0\\ 0&84&82&-99&-79&117&26&-97&-30&-5&1&0\\ 0&-74&0&81&0&12&0&35&0&-55&0&1 \end{pmatrix}$$

RNS(253, 251, 247) has the dynamic range of [-7842620, +7842620] being large enough for 8-bit quantized CNN models. The algorithm $F(10 \times 10, 3 \times 3)$

over RNS(253, 251, 247) need 3 element-wise multiplications in 8-bit (accumulation is 32-bit). The implementation can yield up to 2.08× throughput improvement (or Speed-up).

Alternatively, we can compute the Winograd convolution $F(10 \times 10, 3 \times 3)$ over 16-bit RNS(4001, 4331) for instance.

$$g \circledast d = (A^T_{4001}\Big[[G_{4001}\, g G^T_{4001}] \odot [B^T_{4001}\, d B_{4001}]\Big] A_{4001}\,, \\ A^T_{4331}\Big[[G_{4331}\, g G^T_{4331}] \odot [B^T_{4331}\, d B_{4331}]\Big] A_{4331}) \tag{5}$$

where the transform matrices are

$$A^T_{4001} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 2 & -2 & 3 & -3 & 4 & -4 & 5 & -5 & 0 \\ 0 & 1 & 1 & 4 & 4 & 9 & 9 & 16 & 16 & 25 & 25 & 0 \\ 0 & 1 & -1 & 8 & -8 & 27 & -27 & 64 & -64 & 125 & -125 & 0 \\ 0 & 1 & 1 & 16 & 16 & 81 & 81 & 256 & 256 & 625 & 625 & 0 \\ 0 & 1 & -1 & 32 & -32 & 243 & -243 & 1024 & -1024 & -876 & 876 & 0 \\ 0 & 1 & 1 & 64 & 64 & 729 & 729 & 95 & 95 & -379 & -379 & 0 \\ 0 & 1 & -1 & 128 & -128 & -1814 & 1814 & 380 & -380 & -1895 & 1895 & 0 \\ 0 & 1 & 1 & 256 & 256 & -1441 & -1441 & 1520 & 1520 & -1473 & -1473 & 0 \\ 0 & 1 & -1 & 512 & -512 & -322 & 322 & -1922 & 1922 & 637 & -637 & 1 \end{pmatrix}; \quad G_{4001} = \begin{pmatrix} 222 & 0 & 0 \\ 185 & 185 & 185 \\ 185 & -185 & 185 \\ -1609 & 783 & 1566 \\ -1609 & -783 & 1566 \\ 897 & -1310 & 71 \\ 897 & 1310 & 71 \\ 1533 & -1870 & 522 \\ 1533 & 1870 & 522 \\ -1047 & -1234 & 1832 \\ -1047 & 1234 & 1832 \\ 0 & 0 & 1 \end{pmatrix}$$

$$B^T_{4001} = \begin{pmatrix} -1604 & 0 & -1071 & 0 & -357 & 0 & -1023 & 0 & 55 & 0 & -1 & 0 \\ 0 & -1604 & -1604 & 1326 & 1326 & 969 & 969 & -54 & -54 & 1 & 1 & 0 \\ 0 & 1604 & -1604 & -1326 & 1326 & -969 & 969 & 54 & -54 & -1 & 1 & 0 \\ 0 & 802 & 401 & 736 & 368 & -1638 & -819 & 102 & 51 & -2 & -1 & 0 \\ 0 & -802 & 401 & -736 & 368 & 1638 & -819 & -102 & 51 & 2 & -1 & 0 \\ 0 & 799 & 1600 & 1510 & 1837 & 1827 & 609 & -138 & -46 & 3 & 1 & 0 \\ 0 & -799 & 1600 & -1510 & 1837 & -1827 & 609 & 138 & -46 & -3 & 1 & 0 \\ 0 & 401 & -900 & 1043 & 1261 & -1596 & -399 & 156 & 39 & -4 & -1 & 0 \\ 0 & -401 & -900 & -1043 & 1261 & 1596 & -399 & -156 & 39 & 4 & -1 & 0 \\ 0 & -1121 & 576 & -99 & -820 & 1365 & 273 & -150 & -30 & 5 & 1 & 0 \\ 0 & 1121 & 576 & 99 & -820 & -1365 & 273 & 150 & -30 & -5 & 1 & 0 \\ 0 & 1604 & 0 & 1071 & 0 & 357 & 0 & 1023 & 0 & -55 & 0 & 1 \end{pmatrix}$$

$$A^T_{4331} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 2 & -2 & 3 & -3 & 4 & -4 & 5 & -5 & 0 \\ 0 & 1 & 1 & 4 & 4 & 9 & 9 & 16 & 16 & 25 & 25 & 0 \\ 0 & 1 & -1 & 8 & -8 & 27 & -27 & 64 & -64 & 125 & -125 & 0 \\ 0 & 1 & 1 & 16 & 16 & 81 & 81 & 256 & 256 & 625 & 625 & 0 \\ 0 & 1 & -1 & 32 & -32 & 243 & -243 & 1024 & -1024 & -1206 & 1206 & 0 \\ 0 & 1 & 1 & 64 & 64 & 729 & 729 & -235 & -235 & -1699 & -1699 & 0 \\ 0 & 1 & -1 & 128 & -128 & -2144 & 2144 & -940 & 940 & 167 & -167 & 0 \\ 0 & 1 & 1 & 256 & 256 & -2101 & -2101 & 571 & 571 & 835 & 835 & 0 \\ 0 & 1 & -1 & 512 & -512 & -1972 & 1972 & -2047 & 2047 & -156 & 156 & 1 \end{pmatrix}; \quad G_{4331} = \begin{pmatrix} 1693 & 0 & 0 \\ 689 & 689 & 689 \\ 689 & -689 & 689 \\ -225 & -450 & -900 \\ -225 & 450 & -900 \\ 457 & 1371 & -218 \\ 457 & -1371 & -218 \\ 1064 & -75 & -300 \\ 1064 & 75 & -300 \\ -1626 & 532 & -1671 \\ -1626 & -532 & -1671 \\ 0 & 0 & 1 \end{pmatrix}$$

$$B^T_{4331} = \begin{pmatrix} 1407 & 0 & 579 & 0 & -1017 & 0 & -1023 & 0 & 55 & 0 & -1 & 0 \\ 0 & 1407 & 1407 & 1986 & 1986 & 969 & 969 & -54 & -54 & 1 & 1 & 0 \\ 0 & -1407 & 1407 & -1986 & 1986 & -969 & 969 & 54 & -54 & -1 & 1 & 0 \\ 0 & 1462 & 731 & 76 & 38 & -1638 & -819 & 102 & 51 & -2 & -1 & 0 \\ 0 & -1462 & 731 & -76 & 38 & 1638 & -819 & -102 & 51 & 2 & -1 & 0 \\ 0 & 469 & 1600 & -2161 & -2164 & 1827 & 609 & -138 & -46 & 3 & 1 & 0 \\ 0 & -469 & 1600 & 2161 & -2164 & -1827 & 609 & 138 & -46 & -3 & 1 & 0 \\ 0 & 731 & -900 & 713 & 1261 & -1596 & -399 & 156 & 39 & -4 & -1 & 0 \\ 0 & -731 & -900 & -713 & 1261 & 1596 & -399 & -156 & 39 & 4 & -1 & 0 \\ 0 & -1451 & 576 & 231 & -820 & 1365 & 273 & -150 & -30 & 5 & 1 & 0 \\ 0 & 1451 & 576 & -231 & -820 & -1365 & 273 & 150 & -30 & -5 & 1 & 0 \\ 0 & -1407 & 0 & -579 & 0 & 1017 & 0 & 1023 & 0 & -55 & 0 & 1 \end{pmatrix}$$

The modulus 4001 and 4331 are both coprime to $\frac{1}{\alpha} = 3628800$. The 16-bit RNS has dynamic range $4001 \times 4331 = 17328331$, which allows the convolution output having the maximum magnitude of $\frac{17328331-1}{2} = 8664165$. The 16-bit RNS(4001,4331) requires two element-wise multiply, therefore it has arithmetic reduction 3.13×, which is better than the 2.08× of 8-bit RNS(253,251,247). But, each element-wise multiplication is of 16-bit op.

6 Fast Convolution via integral arithmetic for Convolutional Neural Networks(CNN)

Unlike the conventional Winograd algorithm, which could benefit to CNN for both network training and inference, the integer version can only apply to inference of the low-precision (e.g. INT8) quantized CNN models. For a qunatized CNN layer, its major computation is the 2D convolution, $g \circledast x$, between $(R \times R \times C \times K)$ weight tensor g and $(B \times W \times H \times C)$ input feature maps x, where $R \times R$ is the filter size, C is the depth, K is the filter count (or output channels), B is the batch number and $W \times H$ is the dimension of the 2D input plane. All elements of g and x are signed integers, e.g. from -128 to 127. Then we can utilize the complexity reduced algorithm, equation 4 or 5 described in section 5 to compute the integer convolution.

We can decompose input x into $M \times M$ patches $\{d_i\}$ i.e. $x = \bigoplus_i d_i$, and apply Winograd algorithm $F(M \times M, R \times R)$ over RNS$(m_0, m_1, .., m_{n-1})$ to each corresponding weight g and patch d_i to compute $g \circledast x$ with the reduced arithmetic as equation 6.

$$
\begin{aligned}
g \circledast x = \bigoplus_{B,K,i} \{ & \sum_C A^T_{m_0}((G_{m_0} g^{(C)(K)} G^T_{m_0}) \odot (B^T_{m_0} d_i^{(C)(B)} B_{m_0}))A_{m_0}, \qquad (6) \\
& \sum_C A^T_{m_1}((G_{m_1} g^{(C)(K)} G^T_{m_1}) \odot (B^T_{m_1} d_i^{(C)(B)} B_{m_1}))A_{m_1}, .., \\
& \sum_C A^T_{m_{n-1}}((G_{m_{n-1}} g^{(C)(K)} G^T_{m_{n-1}}) \odot (B^T_{m_{n-1}} d_i^{(C)(B)} B_{m_{n-1}}))A_{m_{n-1}}\} \\
= \{ & A^T_{m_0}(\bigoplus_{B,K,i} (\sum_C ((G_{m_0} g^{(C)(K)} G^T_{m_0}) \odot (B^T_{m_0} d_i^{(C)(B)} B_{m_0}))))A_{m_0}, \qquad (7) \\
& A^T_{m_1}(\bigoplus_{B,K,i} (\sum_C ((G_{m_1} g^{(C)(K)} G^T_{m_1}) \odot (B^T_{m_1} d_i^{(C)(B)} B_{m_1}))))A_{m_1}, \\
& \quad \cdots, \\
& A^T_{m_{n-1}}(\bigoplus_{B,K,i} (\sum_C ((G_{m_{n-1}} g^{(C)(K)} G^T_{m_{n-1}}) \odot (B^T_{m_{n-1}} d_i^{(C)(B)} B_{m_{n-1}}))))A_{m_{n-1}}\}
\end{aligned}
$$

In eq. 7, the forward Winograd Transform of filter e.g. $G_{m_0} w^{(C)(K)} G^T_{m_0}$ can be pre-calculated. The forward transform of input e.g. $B^T_{m_0} x_i^{(C)(B)} B_{m_0}$ is shared or reused across K filters, therefore their computation cost got amortized by factor K. The backward transform was performed after the reduction across depth C due to linearity of transform, so the backward transform was amortized by factor of C.

The point-wise multiply terms in eq. 7, for instance,

$$
\bigoplus_{B,K,i} (\sum_C ((G_{m_0} g^{(C)(K)} G^T_{m_0}) \odot (B^T_{m_0} d_i^{(C)(B)} B_{m_0}))) \pmod{m_0} \qquad (8)
$$

(eq. 8) is a matrix multiply (GEMM) function essentially followed by a modulo operation, which can be executed by existing highly optimized GEMM library,

such as gemmlowp [3] or accelerator. Notably, we can perform the modulo operation after the GEMM to reduce its overhead. In the final step after the backward transform, we convert the $g \circledast x$ from the RNS presentation to the standard format using MRC or CRT.

7 Performance Analysis

The performance of RNS based Winograd convolution depends on the transformation and filter size i.e. $N = M + R - 1$ and R respectively. When computation is carried out in RNS and the cost of Winograd transformation and MRC are amortized due to reuse, the theoretical arithmetic reduction is given by

$$\frac{M^2R^2}{N^2} \times \frac{1}{n}$$

where n is the modulus number of the RNS. Table 2 contains the complexity reduction for different algorithms, $F(M \times M, R \times R)$. The Winograd algorithm has better complexity reduction for large values of M and achieves more benefit for 5×5 filters than the 3×3. Moreover, 2-residue RNS, such as RNS(4001,4331), has more arithmetic reduction than 3-residue case. For example, $F(12 \times 12, 5 \times 5)$ over RNS(4001,4331) generates 7.03× reduction vs 4.69× over RNS(251,241,239). However, 2-residue RNS(4001,4331) requires 16-bit GEMM operation, which will be less efficient than the 8-bit case regarding throughput and power consumption.

Table 2. Complexity reduction of Winograd convolution in RNS.

Algorithms	Arithmetic Complexity Reduction	
$F(M \times M, R \times R)$	RNS(4001,4331)	RNS(251,241,239)
$F(2 \times 2, 3 \times 3)$	1.125×	~~0.75×~~
$F(4 \times 4, 3 \times 3)$	2.00×	1.33×
$F(6 \times 6, 3 \times 3)$	2.53×	1.69×
$F(8 \times 8, 3 \times 3)$	2.88×	1.92×
$F(8 \times 8, 5 \times 5)$	5.56×	3.70×
$F(9 \times 9, 3 \times 3)$	3.01×	2.01×
$F(9 \times 9, 5 \times 5)$	5.99×	3.99×
$F(10 \times 10, 3 \times 3)$	3.13×	2.08×
$F(10 \times 10, 5 \times 5)$	6.38×	4.25×
$F(11 \times 11, 3 \times 3)$	3.22×	2.14×
$F(11 \times 11, 5 \times 5)$	6.72×	4.48×
$F(12 \times 12, 3 \times 3)$	3.31×	2.20×
$F(12 \times 12, 5 \times 5)$	7.03×	4.69×
$F(14 \times 14, 3 \times 3)$	3.45×	2.30×

Our RNS approach is in favor of large transformation, such as 10×10 to 16×16 etc. since the numerical issue is mitigated by using RNS. However, the

[3] https://github.com/google/gemmlowp

Table 3. Throughput (GOPS) of 8-bit, 16-bit and 32-bit GEMM (32-bit output) on 1 CPU of Arm Cortex-A73. e.g. $1024 \times 64 \times 1024$ GEMM indicates the matrix multiply of 1024×64 by 64×1024.

GEMM	8-bit GOPS	16-bit GOPS
1024×64×1024	11.1	8.46
1024×128×1024	13.3	10.1
1024×256×1024	14.8	10.9
256×256×256	14.5	11.1
512×512×512	15.4	11.2
1024×1024×1024	14.6	9.58
2048×2048×2048	14.2	11.2
4096×4906×4096	14.5	9.83

computation cost of Winograd transform, both forward and backward ones, will be higher than using small transformation.

The critical path of the computation is the element-wise multiplication, which is low-precision GEMM operations. Table 3 shows the throughput in GOPS(Giga (10^9) Operations Per-Second) of 8-bit and 16-bit GEMM measured on a single core of Arm Cortex-A73 CPU for variety of size and shape.

For a given hardware e.g. CPU, GPU or accelerator we can determine the optimal implementation based on table 2 and the corresponding GEMM performance. For example, targeting Arm Cortex-A73 CPU used in the benchmark, if we choose RNS(4001,4331) to compute the convolution using $F(12 \times 12, 5 \times 5)$ with $1024 \times 1024 \times 1024$ GEMM, it will have a theoretical speed-up up to $\frac{7.03 \times 9.58}{14.6} = 4.6\times$ to the Im2col+INT16GEMM baseline, while the improvement of RNS(253,251,247) is about $\frac{4.69 \times 14.6}{14.6} = 4.69\times$ over Im2col+INT8GEMM. So, RNS(251,241,239) and RNS(4001,4331) happen to deliver roughly the same improvement with the benchmark program on the Cortex-A73 CPU specifically, but in general 8-bit implementation RNS(251, 241, 239) will consume less power since it uses 8-bit arithmetic. Other hardware, for example Nvidia's RTX2020Ti GPU with up to 215 TOPS of INT8 ops [4], could potentially gain up to a factor of $2.30\times$ or $4.69\times$ performance boost for 3×3 or 5×5 filters respectively through 16×16 RNS-Winograd transformation.

8 Experiments

To validate the proposal, the RNS based Winograd convolution algorithm was implemented in a highly optimized kernel in C on Ubuntu Linux. The program takes advantage of ILP (vector units) to boost the throughput of Winograd transforms, MRC and GEMM functions.

The 2D convolution of 8-bit quantized (for both weight and activation) VGG16 network was benchmarked using the RNS based Winograd algorithm implemented on Arm Cortex-A73 CPU. The convolution output of all CNN layers are within the range of $[-3.0 \times 10^5, 3.0 \times 10^5]$ measured from validation images of

[4] https://devblogs.nvidia.com/nvidia-turing-architecture-in-depth/

Table 4. Inference performance of 8-bit activation and 8-bit weight quantized CNN layers of VGG16 with Winograd algorithm $F(14 \times 14, 3 \times 3)$ over RNS(251,241,239) and RNS(4001,4331) on Arm Cortex-A73, having 71.4% top-1 prediction accuracy with ImageNet dataset. The corresponding transforms are in the supplementary materials. The speed-up of RNS(251,241,239) and RNS(4001,4331) are the runtime improvement relative to the standard INT8 and INT16 Im2col+GEMM convolution baselines respectively.

VGG16 model	conv2d op (int8) x (int8)	Winograd Algorithm	Speed-up RNS(251,241,239)	Speed-up RNS(4001,4331)
conv1_1	$(224, 224, 3) \times (3, 3, 3, 64)$	-†	1×	1×
conv1_2	$(224, 224, 3) \times (3, 3, 3, 64)$	$F(14 \times 14, 3 \times 3)$	1.86×	2.05×
conv2_1	$(112, 224, 64) \times (3, 3, 64, 64)$	$F(14 \times 14, 3 \times 3)$	1.97×	2.13×
conv2_2	$(112, 112, 64) \times (3, 3, 64, 128)$	$F(14 \times 14, 3 \times 3)$	2.07×	2.25×
conv3_1	$(56, 56, 128) \times (3, 3, 128, 128)$	$F(14 \times 14, 3 \times 3)$	2.14×	2.33×
conv3_2	$(56, 56, 128) \times (3, 3, 128, 256)$	$F(14 \times 14, 3 \times 3)$	2.15×	2.37×
conv3_3	$(56, 56, 256) \times (3, 3, 256, 256)$	$F(14 \times 14, 3 \times 3)$	2.16×	2.35×
conv4_1	$(28, 28, 256) \times (3, 3, 256, 512)$	$F(14 \times 14, 3 \times 3)$	2.21×	2.40×
conv4_2	$(28, 28, 512) \times (3, 3, 512, 512)$	$F(14 \times 14, 3 \times 3)$	2.25×	2.37×
conv4_3	$(28, 28, 512) \times (3, 3, 512, 512)$	$F(14 \times 14, 3 \times 3)$	2.27×	2.39×
conv5_1	$(14, 14, 512) \times (3, 3, 512, 512)$	$F(14 \times 14, 3 \times 3)$	2.21×	2.44×
conv5_2	$(14, 14, 512) \times (3, 3, 512, 512)$	$F(14 \times 14, 3 \times 3)$	2.24×	2.39×
conv5_3	$(14, 14, 512) \times (3, 3, 512, 512)$	$F(14 \times 14, 3 \times 3)$	2.22×	2.43×
average			2.02×	2.20×

† Fallback to the baseline.

ImageNet dataset. We used RNS(251,241,239) and RNS(4001,4331), which have the large enough dynamic ranges, $[-7228674, 7228674]$ and $[-8664165, 8664165]$ respectively to guarantee the correctness of the computation.

Using algorithm $F(14 \times 14, 3 \times 3)$, the performance improvement or speed-up over the Im2col+INT8/16 GEMM baselines for both 8-bit and 16-bit RNS are listed in table 4. The overall convolution computation latency reduction is **2.02**× for 8-bit RNS(251,241,239) or **2.20**× for 16-bit RNS(4001,4331). On average, the execution overheads, measured in time, of the 8-bit RNS(251, 241, 239) are 7.9% for the forward Winograd Transform of input feature maps, 9.2% for the backward Winograd transform of output, and 1.1% for MRC while for the 16-bit RNS(4001, 4331), the corresponding overheads are 9.4%, 10.2%, and 1.3% respectively. Table 5 provides extra experimental results for 8-bit ResNet50-v1 and Inception v1 and v3 models using INT8 arithmetic ops. Notably, the Inception-v3 contains three 5×5 convolutional layers, (1) Mixed_5/Branch_1/Conv2d_0b_5x5, (2) Mixed_5c/Branch_1/Conv_1_0c_5x5 and (3) Mixed_5d/Branch_1/Conv2d_0b_5x5 with $(5 \times 5 \times 48 \times 64)$ kernels. The average speed-up for the 5×5 layers are 2.31× with 8-bit 3-residue RNS.

9 Conclusions

We proposed a Residue Number System (RNS) based fast integral Winograd convolution that overcomes the computational instability of the conventional Winograd algorithm. The method enables the execution of the Winograd algorithm using low cost, low precision arithmetic operations (e.g. INT8 MAC)

Table 5. Inference performance improvement over the Im2col+INT8GEMM baseline of CNN layers for 8-bit quantized ResNet50-v1, Inception v1 and v3 models with ImageNet dataset, using 8-bit RNS(251,241,239).

Models	Bits of weight/input	Top-1 Acc.(%)	Speed-up of CNN layers[†]
ResNet50-v1	8/8	75.1	1.76×
Inception-v1	8/8	70.1	1.82×
Inception-v3	8/8	77.5	1.35×

[†] Not include the CNN layers with the stride ≥ 2.

for inference of existing quantized CNN networks. The convolution outputs are precise, which means there is no prediction accuracy degradation with the RNS-based Winograd convolution scheme we have presented.

Our RNS-based approach can benefit the common hardware platforms, including CPU, GPU, and hardware accelerators, which can deliver high throughput, low cost integer MAC operations. The theoretical performance improvement of 8-bit quantized CNN layers can be up to 2.3× and 4.6× over 8-bit 3-residue RNS for 3×3 and 5×5 CNN layers respectively using up to 16×16 transformation.

The experiment showed, on average, the new proposal improved the runtime performance of 3×3 INT8 CNN layers by 2.02× using power efficient 8-bit arithmetic and 2.20× for 16-bit arithmetic over the standard Im2col + INT8 and INT16 GEMM baseline performances respectively measured on an Arm Cortex-A73 mobile CPU using the 8-bit quantized VGG16 model, including the computation overheads such as Winograd transforms over RNS, modulo, and MRC operations etc. The new proposal achieved higher improvement e.g. 2.31× for the CNN layers with larger filter size i.e. 5×5 in Inception-v3.

Although it is possible to increase the transformation size (i.e. $> 16 \times 16$), to further boost arithmetic reduction, the transformation cost increases roughly linearly, therefore it is a reasonable trade-off to choose transformation size from 8 to 16.

References

1. Barabasz, B., Anderson, A., Soodhalter, K.M., Gregg, D.: Error Analysis and Improving the Accuracy of Winograd Convolution for Deep Neural Networks. arXiv e-prints arXiv:1803.10986 (Mar 2018)
2. Courbariaux, M., Bengio, Y.: Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. CoRR **abs/1602.02830** (2016)
3. Dally, W.: Nips tutorial 2015. https://media.nips.cc/Conferences/2015/tutorial slides/Dally-NIPS-Tutorial-2015.pdf (2015)
4. Knuth, D.E.: The Art of Computer Programming, vol. Volume 1: Fundamental Algorithms (3rd ed.) §1.2.3: Sums and Products: Exercise 40 (1997)
5. Knuth, D.E.: The Art of Computer Programming, vol. Volume 2: Seminumerical Algorithms (Third ed.) Section 4.3.2 (pp. 286–291), exercise 4.6.2–3 (page 456). Addison-Wesley (2001)
6. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (May 2017). https://doi.org/10.1145/3065386, http://doi.acm.org/10.1145/3065386
7. Lavin, A., Gray, S.: Fast Algorithms for Convolutional Neural Networks. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
8. Liu, X., Pool, J., Han, S., Dally, W.J.: Efficient sparse-winograd convolutional neural networks. In: International Conference on Learning Representations (2018), https://openreview.net/forum?id=HJzgZ3JCW
9. Liu, Z.G., Mattina, M.: Learning low-precision neural networks without straight-through estimator (ste). In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. pp. 3066–3072. International Joint Conferences on Artificial Intelligence Organization (7 2019). https://doi.org/10.24963/ijcai.2019/425, https://doi.org/10.24963/ijcai.2019/425
10. Mathieu, M., Henaff, M., LeCun, Y.: Fast training of convolutional networks through ffts. In: Bengio, Y., LeCun, Y. (eds.) 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings (2014)
11. Meng, L., Brothers, J.: Efficient Winograd Convolution via Integer Arithmetic. arXiv e-prints arXiv:1901.01965 (Jan 2019)
12. Mohan, P.V.: Residue Number Systems: Algorithms and Architectures. Kluwer Academic Publishers (2002)
13. Schonheim, J.: Conversion of Modular Numbers to their Mixed Radix Representation by Matrix Formula . Mathematics of Computation, pp. 253-257 (1967)
14. Winograd, S.: Arithmetic complexity of computations, vol. 33. Siam (1980)