Neurocomputing 460 (2021) 9–19

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

A tile-fusion method for accelerating Winograd convolutions ☆

Zeyu Ji, Xingjun Zhang *, Zheng Wei, Jingbo Li, Jia Wei

School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China

ARTICLE INFO

Article history:
Received 13 August 2020
Revised 29 April 2021
Accepted 3 June 2021
Available online 5 June 2021
Communicated by Zidong Wang

Keywords:
Fast convolution
Winograd algorithm
Parallel algorithm
Convolutional neural network
Tile size
Zero padding

ABSTRACT

Compared with fast convolution methods such as im2col and the fast Fourier transform, Winograd-based convolution, which has been widely applied to accelerate convolutional neural networks (CNNs), can provide high performance with smaller filters. Although there are several reported studies on the algorithmic optimization of CNNs, most of them are targeted at hardware architectures. The existing implementations of the Winograd method perform well below what one would expect, due to the fact that the tile size of Winograd-based convolution is usually empirical and the features of each convolution layer are ignored. This study aims to fill this gap and focuses on the efficient implementation of Winograd-based convolution in the CNN model. Specifically, we discuss the causes of poor performance, calculate the coefficient of computation complexity model and demonstrate a speedup in the inference process using an elaborate tile-fusion method, which derives the optimal tile size for each convolution layer in a CNN model. Compared with the representative existing implementations of CuDNN with a 4×4 tile, Arm Compute Library with a 6×6 tile, and NNPACK with an 8×8 tile, the results show significant performance improvements on of up to $1.89\times$, $1.29\times$ and $1.17\times$, respectively.

1. Introduction

A convolutional neural network(CNN) [1] is a set of deep learning algorithms that have achieved notable performance for diversiform AI tasks, including video surveillance, speech recognition, natural language processing, and self-driving. In the past decade, CNNs have presented a grander prospect and have been the focus of a great deal of research. Convolutional layer, which is memory intensive and computationally expensive, is prevalent in advanced CNNs, including AlexNet, VGG, OverFeat, and ResNet. Consequently, they dominate the total performance of a CNN.

The exponential growth of data and increasingly complex models make the diffusion of deep learning possible. As well, they lead to an additional training overhead and extra computation time. Therefore, training and inferring CNN models in a short time is a tremendous challenge for researchers.

Deep neural networks(DNNs) can be trained in a short time by using distributed training techniques on clusters equipped with huge memory resources and computational power [2–4]. In a distributed environment, a variety of consistency strategies including bulk synchronous parallel (BSP) [5,6], staleness synchronous parallel(SSP) [7,8], and asynchronous parallel(ASP) [8,9] are proposed allowing us to train and serve a model on multiple physical machines. To guarantee the same computational results as on a single machine, BSP forces all the slave workers to globally synchronize in each iteration. ASP [8,9] relaxes the consistency constraint and allows every worker to update the global parameters as soon as the worker finishes its own iteration. As a tradeoff between BSP and ASP, the SSP [7,8] introduces a new parameter, called the staleness threshold, which bounds iteration cycles between the fastest worker and the slowest worker in a cluster. Within the staleness threshold, workers can update themselves instead of pulling global parameters from the server via the network.

Compared to distributed training, the training and inference of a CNN on a single machine is limited by how efficiently the I/O bandwidth can be occupied. The model compressing techniques, which alleviate the I/O problem, aid in the developing smaller and more efficient neural networks, including parameter pruning, parameter sharing, low-rank decomposition, and knowledge distillation. They achieve DNN compression and acceleration without significantly degrading model performance. By removing redundancy and low information weights from existing well-trained deep network models, parameter pruning-based approaches [10,11] reduce the parameters of neural networks and speed up the computation time. Parameter sharing involves the design of mapping to share multiple parameters with the same data. In recent years, quantiza-

☆ This work was supported by the National Key Research and Development Program of China (2016YFB0200902).

* Corresponding author.
E-mail address: xjzhang@xjtu.edu.cn (X. Zhang).

https://doi.org/10.1016/j.neucom.2021.06.003
0925-2312/

tion [12] has been widely used as a parameter sharing technique by using low bit-width numbers instead of floating-point numbers. Additionally, hash functions and structured linear mapping can be used as a form of parameter sharing. The main idea of low-rank decomposition [13] is to use matrix or tensor decomposition technique to estimate and decompose the original convolution kernel in a DNN model. Nevertheless, these methods are only applicable to neural networks that are overparameterized and where sufficient redundant information exists that can be safely reduced without significant loss of accuracy. The studies in [14,15] propose a more efficient network architecture which generates acceptable accuracy with a relatively small model size.

A number of optimization approaches have been proposed to reduce the overhead of convolution algorithms, such as sliding windows, im2col, fast Fourier transforms (FFTs), and Winograd algorithms. A sliding window [16,17] is a spatial domain-based algorithm that sums the product of a feature map with a corresponding filter kernel to generate the output data. Convolution based on img2col [18,19] can be considered as fast matrix–matrix multiplication, and profits from highly optimized linear algebra packages, e.g., BLAS and MKL. FFT-based convolution [20,21] uses the following principle: multiplication in the frequency domain corresponds to convolution in the time domain. The input signal is transformed into the frequency domain using the discrete Fourier transform (DFT), multiplied by the frequency response of the filter, and then transformed back into the time domain using the inverse DFT. Winograd-based convolution [22], is developed for fast computation of finite impulse response (FIR) filters [23]. The Hadamard product in the Winograd domain corresponds to convolution in the spatial domain, which is similar to FFT convolution. FFT and Winograd methods fundamentally reduce the computational complexity of the convolution algorithm, and are broadly used for accelerating CNN models.

Winograd-based convolution is widely integrated into numerous off-the-shelf deep learning frameworks [17,18,24,25], due to state-of-art performance for CNNs with small filters. However, most implementations of Winograd based convolution are only optimized for parallel computation on the different platforms, including CPUs [26,27], GPUs [18], MICs [28], FPGAs [29,30], and ASIC devices [31]. The DNN community has witnessed very little research on improving the performance of the Winograd-based convolution. Theoretically, Winograd-based convolution should speed up the process of computation, however, in practice, several existing challenges make it difficult to fully utilize the computational resources or memory bandwidth. To minimizing the number of necessary calculations, the existing implementations of Winograd convolution [22,27] need to hand-craft the minimal transform of the input data, the kernel, and the output data. Hence, it is difficult to offer a flexible tile size for each convolutional layer. For instance, Nvidia CuDNN has a fixed tile size of 4×4, the Arm Compute Library has a fixed tile size of 6×6, and NAPACK has a fixed tile size of 8×8. Taking a kernel size of 3×3 as an example, when the output tile size is 4×4, the tile size of Winograd-based convolution is 4×4(m-r + 1). In the first convolution layer of VGG, the output size is 222(224–3 + 1), which is not divisible by 4. To determine the indivisibility problem, the padding method has been introduced; however, it degrades the performance of Winograd-based convolution. The detrimental effects of these problems are more evident in resource-constrained applications, e.g., cell phones, smart wearables, and vehicular devices. The small tile cannot scale the performance of Winograd-based convolution. Instead, the large tile introduces zero-padding, which increases the overhead at both the transformation and the matrix multiplication stages. Previous studies [28,27] have only proposed experimental tile sizes for Winograd transforms, and they do not analyze the arithmetic complexity of Winograd-based convolution quantitatively.

To address this issue, a tile-fusion method is proposed in this paper to ameliorate the performance of Winograd-based convolution. We attempt to decompose the construction of Winograd transforms and establish a quantitative arithmetic complexity model that can calculate the computational complexity by the features of each convolutional layer (e.g., kernel size (R), channels (C), width (W), height (H), number of kernels (K)). We then select an optimal tile size for each convolution layer based on the model. The main contributions of this work are summarized as follows.

1. A comprehensive model analysis of Winograd-based convolution is presented.
2. A quantitative arithmetic complexity model for Winograd-based convolution is generated. The parameters include kernel size (R), channels (C), width (W), height (H), and number of kernels (K).
3. Base on the above model, a tile-fusion method is proposed to minimize the total overhead of a CNN.

The remainder of this paper is organized as follows: Section 2 covers related work; Section 3 provides background material; Section 4 presents how to calculate the coefficient of computation complexity model; Section 5 describes the optimization method; Section 6 summarizes the experiment; and Section 7 concludes the study.

2. Related work

2.1. Direct convolution

The sliding window algorithm was first applied in Caffe [17], yet it does not offer satisfying performance. For reducing the training time of a CNN, Fang optimized sliding window-based convolution on the Sunway TaihuLight supercomputer [16]. Nvidia integrated im2col into CuDNN, which is a GPU accelerated library of primitives for DNNs. Cho and Brand [19] improved the im2col method, whose main idea is to lower the input matrix in a highly compact way to improve the performance of computing a convolution in a CNN. Meanwhile, the im2col method incurs large memory overhead due to the redundancy of the transformation of the input data, which is not friendly for a device having a relatively small cache and low bandwidth.

2.2. Fast convolution

An important class of algorithms referred to as fast convolution includes the FFT and Winograd methods. FFT-based [21,20] convolution reduces the arithmetic complexity of convolutional layers by using the Fourier transform, which is employed in GPUs and CPUs. However, it is inefficient when the kernels are relatively small. More recently, Lavin et al. proposed Winograd-based convolution [22], which can dramatically reduce the number of multiplications in convolution and demonstrated that it can achieve better performance than the FFT-based method for small kernels (e.g., $3\times$). The general recipe of Winograd-based convolution is composed of three parts. Both the data and the kernel are first transformed to the Winograd domain. Next, an element-wise multiplication is performed instead of sliding-window convolution in the spatial domain. An inverse transformation generates the result of the convolution. GPU implementations [18] have been developed by Nvidia and Nervana Systems. Open source libraries, such as NNPACK [25] and Intel MKL-DNN [32] integrate CPU implementations. Furthermore, Budden et al. [26] and Jia et al. [28] independently generated a method for arbitrary dimensions and kernel sizes on a CPU

and an Intel Xeon Phi. Shen et al. [29] and Shi et al. [30]have provided FPGA implementations. Some studies [31,27] optimize the Winograd-based convolution for mobile phones. However, all current Winograd-based implementations cannot adjust the efficient tile-size of the Winograd transform for each convolutional layer in a modern ConvNet due to manual handcrafted minimal algorithms. Thus, for ConvNet, a fixed tile-size for the Winograd transform means that the convolution layers will acquire disparity inefficiency because of their own features. Meanwhile, it will also yield massive memory access and a padding problem.

3. Background

3.1. One-Dimensional Winograd-Based Convolution

Shmuel Winograd introduced an algorithm for fast convolution of FIR filters in the signal processing domain [23].

We denote the computation of m outputs of an r-tap FIR filter by $F(m,r)$, which requires $m \times r$ multiplications for direct convolution. For the Winograd-based method, the requirement for the number of multiplications is reduced to $m+r-1$. Taking $F(2,3)$ as an example, the Winograd-based method first transforms the input data $d=(d_0,d_1,d_2,d_3)$ and filter data $g=(g_0,g_1,g_2)$ to $j=(j_0,j_1,j_2,j_3)$ and $h=(h_0,h_1,h_2)$ sparately via

$$\begin{aligned} j_0 &= d_0 - d_2, h_0 = g_0 \\ j_1 &= d_1 + d_2, h_1 = \tfrac{g_0+g_1+g_2}{2} \\ j_2 &= d_2 - d_1, h_2 = \tfrac{g_0-g_1+g_2}{2} \\ j_3 &= d_1 - d_3, h_3 = g_2 \end{aligned} \tag{1}$$

Next, it performs the Hadamard product with j and h as follows:

$$\begin{aligned} c_0 &= j_0 \times h_0, c_1 = j_1 \times h_1 \\ c_2 &= j_2 \times h_2, c_3 = j_3 \times h_3 \end{aligned} \tag{2}$$

Lastly, the final result $y=(y_0,y_1)$ is generated through:

$$y_0 = c_0 + c_1 + c_2, y_1 = c_1 - c_2 - c_3 \tag{3}$$

The procedure above can be written in matrix form as $Y=A^T\left[[Gg]\odot\left[B^Td\right]\right]$, where $\odot$ indicates the Hadamard produce. The Hadamard product in $F(2,3)$ requires $m+r-1=2+3-1=4$ multiplications, while a direct convolution requires $m \times r = 2 \times 3 = 6$ multiplications.

$$B^T=\begin{bmatrix}1&0&-1&0\\0&1&1&0\\0&-1&1&0\\0&1&0&-1\end{bmatrix} G=\begin{bmatrix}1&0&0\\\frac{1}{2}&\frac{1}{2}&\frac{1}{2}\\\frac{1}{2}&-\frac{1}{2}&\frac{1}{2}\\0&0&1\end{bmatrix} A^T=\begin{bmatrix}1&1&1&0\\0&1&-1&-1\end{bmatrix} g=[g_0 g_1 g_2]^T d=[d_0 d_1 d_2 d_3]^T \tag{4}$$

3.2. Two-dimensional Winograd-based convolution

The one-dimension Winograd method can be unfolded to two-dimension or higher dimensional convolutions by being nested with itself. $F(m \times m, r \times r)$ can be denoted as follows:

$$Y = A^T\left[\left[GgG^T\right]\odot\left[B^TdB\right]\right]A \tag{5}$$

where vector g is an $r \times r$ filter and vector d is a $l \times l$ ($l=m+r-1$) input tile. The size of the output tile Y is $m \times m$. The nesting method can be also used for non square filters and outputs. Taking $F(2\times 2, 3\times 3)$ as an example, it uses $4\times 4=16$ multiplications, while the direct convolution method requires $2\times 2\times 3\times 3=36$ multiplications. Thus it achieves a $36/16=2.25$ multiplication reduction. The input transform, filter transform and output transform use 32,28, and 24 floating-point instructions, respectively.

3.3. Winograd algorithm in ConvNet layer

$F(m\times m, r\times r)$ can be employed to compute ConvNet layers with $r\times r$ kernels. the input image, each channel is divided into tiles of size $(m+r-1)\times(m+r-1)$ and the neighboring tiles are overlapped with $r-1$ elements, which yields $P=N\lceil H/m\rceil\lceil W/m\rceil$ tiles per channel C. Denoting $U=GgG^T$ and $V=B^TdB$, we have:

$$Y = A^T[U\odot V] \tag{6}$$

Labeling tile coordinates as $(\bar{x},\bar{y})$, we can obtain another form of the formula with image i, filter k, and tile $(\bar{x},\bar{y})$ as follows:

$$\begin{aligned} Y_{i,k,\bar{x},\bar{y}} &= \sum_{c=1}^{C} D_{i,c,\bar{x},\bar{y}} * G_{k,c} \\ &= \sum_{c=1}^{C} A^T\left[U_{k,c}\odot V_{c,i,\bar{x},\bar{y}}\right]A \\ &= A^T\left[\sum_{c=1}^{C} U_{k,c}\odot V_{c,i,\bar{x},\bar{y}}\right]A \end{aligned} \tag{7}$$

Using the benefits from Eq. (7), the inverse transform can be reduced over C channels, which amortizes the cost of the output transform over the number of channels. With the formulation above, the Winograd method is efficiently implemented on CPUs, GPUs and FPGAs. Algorithm 1 is implemented firstly by Lavin A and Gray S [22] on GPU.

Algorithm 1: Compute ConvNet layer with Winograd minimal filtering algorithm $F(m\times m, r\times r)$

$P=N\lceil H/m\rceil\lceil W/m\rceil$ is the number of image tiles.
$\alpha=m+r-1$ is the input tile size.
Neighboring tiles overlap by $r-1$.
$d_{c,b}\in\mathbb{R}^{\alpha\times\alpha}$ is input tile b in channel c.
$g_{k,c}\in\mathbb{R}^{r\times r}$ is kernel k in channel c.
G, B^T and A^T are filter, data, and inverse transforms.
$Y_{k,b}\in\mathbb{R}^{m\times m}$ is output tile b in filter k.
for $k=0$ to K **do**
  for $c=0$ to C **do**
    $u=Gg_{k,c}G^T\in\mathbb{R}^{\alpha\times\alpha}$
    Scatter u to matrices U:$U^{\xi,\nu}_{k,c}=u_{\xi,\nu}$
for $b=0$ to P **do**
  for $c=0$ to C **do**
    $v=B^Td_{c,b}B\in\mathbb{R}^{\alpha\times\alpha}$
    Scatter v to matrices V:$V^{\xi,\nu}_{c,b}=v_{\xi,\nu}$
for $\xi=0$ to α **do**
  for $\nu=0$ to α **do**
    $M^{\xi,\nu}=U^{\xi,\nu}V^{\xi,\nu}$
for $k=0$ to K **do**
  for $b=0$ to P **do**
    Gather m from matrices M:$m_{\xi,\nu}=M^{\xi,\nu}_{k,b}$
    $Y_{\xi,\nu}=A^TmA$

4. Methodology

4.1. Arithmetic comlexity analysis

Algorithm 1 can be decomposed into four parts:

1. Filter transform;
2. Input tiles transform;

3. General matrix to matrix multiplication (GEMM);
4. Inverse transform.

Compared with direct convolution, the Winograd-based method reduces the requirement of the number of multiplications in part3, while it attaches overhead of the transforms in parts 1, 2, and 4. For $F(m\times n, R\times S)$, the arithmetic complexity of the multiplication stage in a one layer of a ConvNet is given by

$$X = N\left\lceil\frac{H}{m}\right\rceil\left\lceil\frac{W}{n}\right\rceil CK(m+R-1)(n+S-1) \tag{8}$$

When $m=n=1, X$ equals the computational complexity of direct convolution. As a result, direct convolution in Winograd-based form is $F(1\times 1, R\times S)$. To simplify the model, we assume that W is divisible by m and H is divisible by n. We also assume that filters and input tiles are squared ($R=S, m=n$). Then the Eq. (8) can be rewritten as follows:

$$X = \frac{(m+R-1)^2}{m^2} N\times H\times W\times C\times K \tag{9}$$

We denote $\alpha = (m+R-1)^2 = tile\_size^2$, which is the arithmetic complexity of multiplication per input tiles. Idem, the arithmetic complexities of the data, filter, and inverse transforms can be written as

$$\begin{aligned} T(D) &= \tfrac{\beta}{m^2} NHWC \\ T(F) &= \gamma CK \\ T(I) &= \tfrac{\delta}{m^2} NHWC \end{aligned} \tag{10}$$

where β, γ, and δ are the number of floating instructions used by the input, filter, and inverse transforms, respectively, for a single tile. Dividing $T(D), T(F)$, and $T(I)$ by X generates therelative formulas. The terms β', γ', and δ' denote the normalized computational complexity of the input, filter, and inverse transforms, separately. Further, $P = NHW/m^2$ is the number of tiles in each channel, and $\alpha' = \alpha/m^2$ denotes the normalized computational complexity of multiplication. Thus,

$$\begin{aligned} \tfrac{T(D)}{X} &= \tfrac{\beta}{K\alpha} = \tfrac{\beta'}{K} \\ \tfrac{T(F)}{X} &= \tfrac{\gamma}{\frac{NHW}{m^2}\alpha} = \tfrac{\gamma}{P\alpha} = \tfrac{\gamma'}{P} \\ \tfrac{T(I)}{X} &= \tfrac{\delta}{C\alpha} = \tfrac{\delta'}{C} \end{aligned} \tag{11}$$

Summing the terms for the four parts yields the total arithmetic complexity for one ConvNet layer:

$$L = \alpha'\left(1 + \frac{\beta'}{K} + \frac{\gamma'}{P} + \frac{\delta'}{C}\right) N\times H\times W\times C\times K \tag{12}$$

In Eq. (12), we can conclude that α', β', γ', and δ' should be as small as possible for achieving a large speedup. In the case of direct convolution, $\alpha' = \alpha = R$ and $\beta' = \gamma' = \delta' = 0$, hence the theoretical maximum speedup of the Winograd-based method is R^2/α'.

4.2. Transform matrices

According to the definitions of α', β', γ', and δ' in Section 4.1, we can calculate the terms as follows:

$$\begin{aligned} \alpha' &= \tfrac{\alpha}{m^2} = \left(\tfrac{m+R-1}{m}\right)^2 \\ \beta' &= \beta/\alpha \\ \gamma' &= \gamma/\alpha \\ \delta' &= \delta/\alpha \end{aligned} \tag{13}$$

We take $F(4,3)$ as an example to explore how to generate the transform matrices A, B, and G. The three-element filter g and the four-elements signal d can be represented as polynomials:

$$\begin{aligned} g(x) &= g_2x^2 + g_1x + g_0 \\ d(x) &= d_3x^3 + g_2x^2 + d_1x + d_0 \end{aligned} \tag{14}$$

And the linear convolution $g * d$ is equal to the coefficients of the polynomial product

$$y(x) = g(x)d(x) \tag{15}$$

For polynomial $m(x)$ of degree 5, this is equal to

$$y(x) = g(x)d(x)\mathrm{mod}m(x) + R_{m(x)}[y(x)] \tag{16}$$

where $R_{m(x)}[y(x)]$ is the remainder of $y(x)$ divided by $m(x)$. We choose

$$\begin{aligned} m(x) &= m^{(0)}(x)m^{(1)}(x)m^{(2)}(x)m^{(3)}(x)m^{(4)}(x)m^{(5)}(x) \\ &= x(x-1)(x+1)(x-2)(x+2)(x-\infty) \end{aligned} \tag{17}$$

which use the convention of writing $x-\infty$ instead of $R_{m(x)}[y(x)]$. The residues of $g(x)$ and $c(x)$ with respect to $m^{(i)}(x)$ are

$$\begin{aligned} g^{(0)}(x) &= g(x)\mathrm{mod}m^{(0)} = g_0 \\ g^{(1)}(x) &= g(x)\mathrm{mod}m^{(1)} = g_0 + g_1 + g_2 \\ g^{(2)}(x) &= g(x)\mathrm{mod}m^{(2)} = g_0 - g_1 + g_2 \\ g^{(3)}(x) &= g(x)\mathrm{mod}m^{(3)} = g_0 + 2g_1 + 4g_2 \\ g^{(4)}(x) &= g(x)\mathrm{mod}m^{(4)} = g_0 - 2g_1 + 4g_2 \\ g^{(5)}(x) &= g(x)\mathrm{mod}m^{(5)} = g_2 \end{aligned} \tag{18}$$

and

$$\begin{aligned} d^{(0)}(x) &= d(x)\mathrm{mod}m^{(0)} = d_0 \\ d^{(1)}(x) &= d(x)\mathrm{mod}m^{(1)} = d_0 + d_1 + d_2 + d_3 \\ d^{(2)}(x) &= d(x)\mathrm{mod}m^{(2)} = d_0 - d_1 + d_2 - d_3 \\ d^{(3)}(x) &= d(x)\mathrm{mod}m^{(3)} = d_0 + 2d_1 + 4d_2 + 8d_3 \\ d^{(4)}(x) &= d(x)\mathrm{mod}m^{(4)} = d_0 - 2d_1 + 4d_2 - 8d_3 \\ d^{(5)}(x) &= d(x)\mathrm{mod}m^{(5)} = d_3 \end{aligned} \tag{19}$$

We can represent the residues $d^{(i)}(x)$ in matrix form as

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 2 & 4 & 8 \\ 1 & -2 & 4 & -8 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{20}$$

Define $M^{(i)}(x) = m(x)/m^{(i)}(x)$, yielding:

$$\begin{aligned} M^{(0)}(x) &= x^4 - 5x^2 + 4 \\ M^{(1)}(x) &= x^4 + x^3 - 4x^2 - 4x \\ M^{(2)}(x) &= x^4 - x^3 - 4x^2 + 4x \\ M^{(3)}(x) &= x^4 + 2x^3 - x^2 - 2x \\ M^{(4)}(x) &= x^4 - 2x^3 - x^2 + 2x \\ m(x) &= x^5 - 5x^3 + 4x \end{aligned} \tag{21}$$

Matrix B is constructed such that column B_i is the coefficients of $M^{(i-1)}(x)$, and column B_6 is the coefficients of m, yielding:

$$B = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & -4 & 4 & -2 & 2 & 4 \\ -5 & -4 & 4 & -1 & -1 & 0 \\ 0 & 1 & -1 & 2 & -2 & -5 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{22}$$

In order to apply the Chinese Remainder Theorem, we should solve for $n^{(i)}(x), N^{(i)}(x)$, such that

$$n^{(i)}(x)m^{(i)}(x) + N^{(i)}(x)M^{(i)}(x) = 1 \tag{23}$$

which yields:

$$\begin{aligned} N^{(0)}(x) &= \tfrac{1}{4} \\ N^{(1)}(x) &= -\tfrac{1}{6} \\ N^{(2)}(x) &= -\tfrac{1}{6} \\ N^{(3)}(x) &= \tfrac{1}{24} \\ N^{(4)}(x) &= \tfrac{1}{24} \end{aligned} \tag{24}$$

Matrix G is constructed by setting row G_i to be equal to the coefficients of $g^{(i-1)}$ multiplied by $N^{(i-1)}$.

In summary, the transform matrices A, B, and G are determined by $m(x)$ and the kernel. The values of the transform matrices are determined by the form of $m(x)$. The size of the transform matrices is determined by the kernel size and the degree of $m(x)$, which equals to value of the tile.

The value of a transform matrix, which is generated by Chinese Remainder Theorem in Lavin's research, affects the arithmetic complexity model. In number theory, the Chinese Remainder Theorem states that if one knows the remainders of the Euclidean division of an integer n by several integers, then one can uniquely determine the remainder of the division of n by the product of these integers, under the condition that the divisors are pairwise coprime. This cannot be generalized to any principal ideal domain, but its generalization to Euclidean domains is straightforward. The univariate polynomials over a field is the typical example of a Euclidean domain, which is not the integers.

The size of matrix B^T is equal to $(m+R-1)^2$ (the tile size). The size of matrix G is equal to $(m+R-1)\times R$. The size of matrix A^T is equal to $m\times(m+R-1)$.

$$\begin{aligned} \beta &= \beta_{1D}\times 2\times(m+R-1) \\ \gamma &= \gamma_{1D}\times(m+R-1+R) \\ \delta &= \delta_{1D}\times(m+R-1+m) \end{aligned} \tag{25}$$

In Eq. (25), $\beta_{1D}, \gamma_{1D}, \delta_{1D}$ are the arithmetic complexities of the input, filter, and inverse transforms respectively in the case of one-dimension.

4.3. Arithmetic complexity in manual mode

In Eq. (5), the input data transform processing is expressed as B^TdB, the filter transform processing is expressed as GgG^T, and the inverse transform processing is expressed as $A^T[\cdots]A$. We use the Chinese Remainder Theorem to generate the matrices B^T, G, A^T, and find that the value of β relies on the transform matrix B, and that the size of matrix B is only determined by the tile size $(m+R-1)$. Similarly the value of γ relies on the transform matrix G, and the size of matrix G is determined by the tile size $(m+R-1)$ and the kernel size R. The value of δ relies on the transform matrix A, and the size of matrix A is determined by the tile size $(m+R-1)$ and the output size m. If the tile size is even, then

$$\begin{aligned} \beta_{1D} &= \tfrac{tile^2-2tile}{2} = tile\times\left(\tfrac{tile}{2}-1\right) \\ \gamma_{1D} &= \tfrac{tile}{2}\times(R+1)-R \\ \delta_{1D} &= \tfrac{tile^2-(R+1)\times tile}{2}+2(R-1) \end{aligned} \tag{26}$$

If the tile size is odd, then

$$\begin{aligned} \beta_{1D} &= (tile-3)^2+\tfrac{3tile}{2}-\tfrac{5}{2} \\ \gamma_{1D} &= \tfrac{tile-1}{2}\times(R+1)-R \\ \delta_{1D} &= \tfrac{tile^2-R\times tile+3R-5}{2} \end{aligned} \tag{27}$$

According the Eq. (25), we can calculate β, γ, and δ. Then we will calculate β', γ', and δ' respectively depending on Eq. (13).

If the tile size is even, then

$$\begin{aligned} \beta' &= tile-2 \\ \gamma' &= \tfrac{R+1}{2}+\tfrac{R^2-R}{2tile}-\tfrac{R^2}{tile^2} \\ \delta' &= tile-\tfrac{3R+1}{2}+\tfrac{R^2+8R-9}{2tile}-\tfrac{2(R-1)^2}{tile^2} \\ \alpha' &= \left(\tfrac{tile}{tile+1-R}\right)^2 \end{aligned} \tag{28}$$

If the tile size is odd, then

$$\begin{aligned} \beta' &= 2tile-9+\tfrac{13}{tile} \\ \gamma' &= \tfrac{R+1}{2}+\tfrac{R^2-2R-1}{2tile}-\tfrac{3R^2+R}{2tile^2} \\ \delta' &= tile-\tfrac{3R-1}{2}+\tfrac{R^2+5R-10}{2tile}-\tfrac{(3R^2-8R+5)}{2tile^2} \\ \alpha' &= \left(\tfrac{tile}{tile+1-R}\right)^2 \end{aligned} \tag{29}$$

Substituting the value of β', γ', and δ' into Eq. (12), the conclusion can be drawn that the relationship the between total arithmetic complexity and the tile size is that of an anti 'S' shape.

4.4. Arithmetic complexity in auto mode

Since the Winograd-based convolution via matrix multiplication can be extended to arbitrary dimensions facilely comparing to manually reducing the transform costs. In some studies [26,28], the Winograd transform and the inverse transform are implemented by matrix multiplication. We can construct the arithmetic complexity model by a similar analysis as that used above.

For one $n\times m$ matrix and one $m\times p$ matrix, the arithmetic complexity of schoolbook matrix multiplication is $n\times m\times p$. In Section 4.2, the sizes of matrices G, B, and A are $(m+R-1)\times R, (m+R-1)^2$, and $(m+R-1)\times m$. Then we can conclude that the total arithmetic complexity of the data, filter, and inverse transforms for the automatic case are as follows:

$$\begin{aligned} \beta &= 2(m+R-1)^3 \\ \gamma &= (m+R-1)R^2+R(m+R-1)^2 \\ \delta &= (m+R-1)m^2+m(m+R-1)^2 \end{aligned} \tag{30}$$

According the definition of β', γ', and δ', we generate the formulas for calculating β', γ', and δ' in auto model as follows:

$$\begin{aligned} \beta' &= \tfrac{\beta}{\alpha} = 2(m+R-1) \\ \gamma' &= \tfrac{\gamma}{\alpha} = R+\tfrac{R^2}{m+R-1} \\ \delta' &= \tfrac{\delta}{\alpha} = m+\tfrac{m^2}{m+R-1} \end{aligned} \tag{31}$$

Substituting the values of β', γ', and δ' into Eq. (12), we can obtain conclusion similar to that of Section 4.3.

By a quantitative analysis of the previous view, when the size of the convolution kernel is fixed, there is a theoretical upper limit in the acceleration effect of the Winograd algorithm, and the speedup of the growth curve is S-shaped. This means that when tile size exceeds a certain threshold value, the acceleration effect will be significantly reduced. Fig. 1 shows the actual reduction in computations as a function of kernels and tile sizes. Fig. 2 shows the actual reduction in computations as a function of channels and kernels. Our optimization is based on the computational complexity model, and we will describe it in the next section.

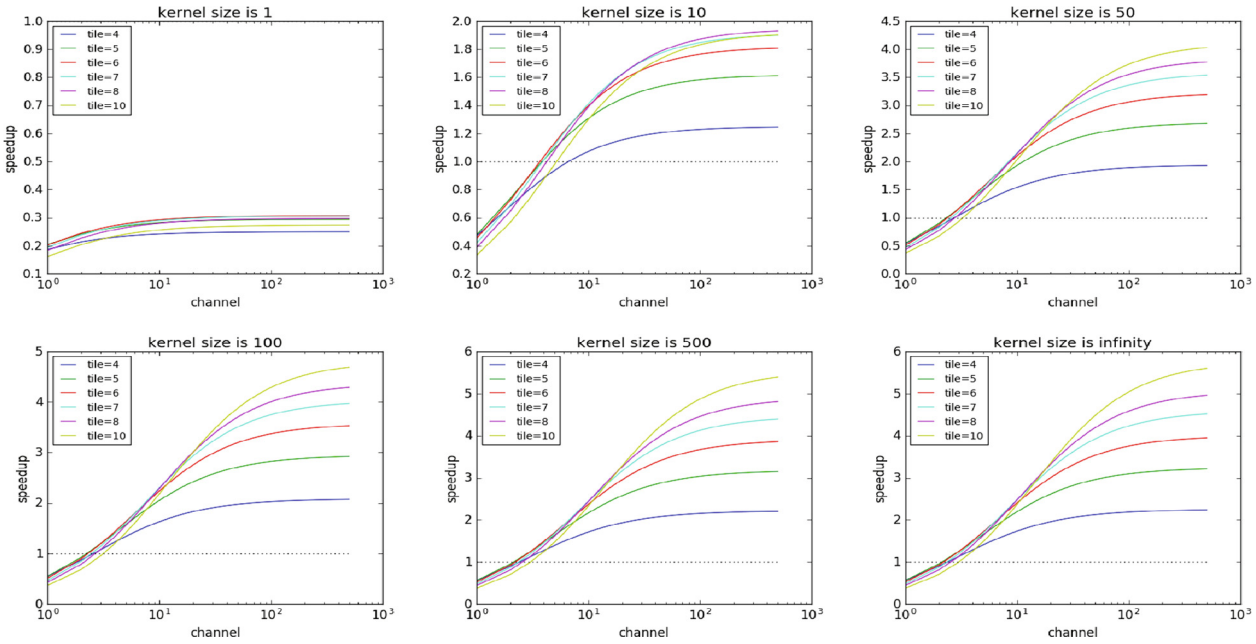

Fig. 1. The speed-up achieved by Winograd convolution for a kernel (3×3) as a function of number layer channels and tile sizes. Dashed line indicates direct convolution baseline.

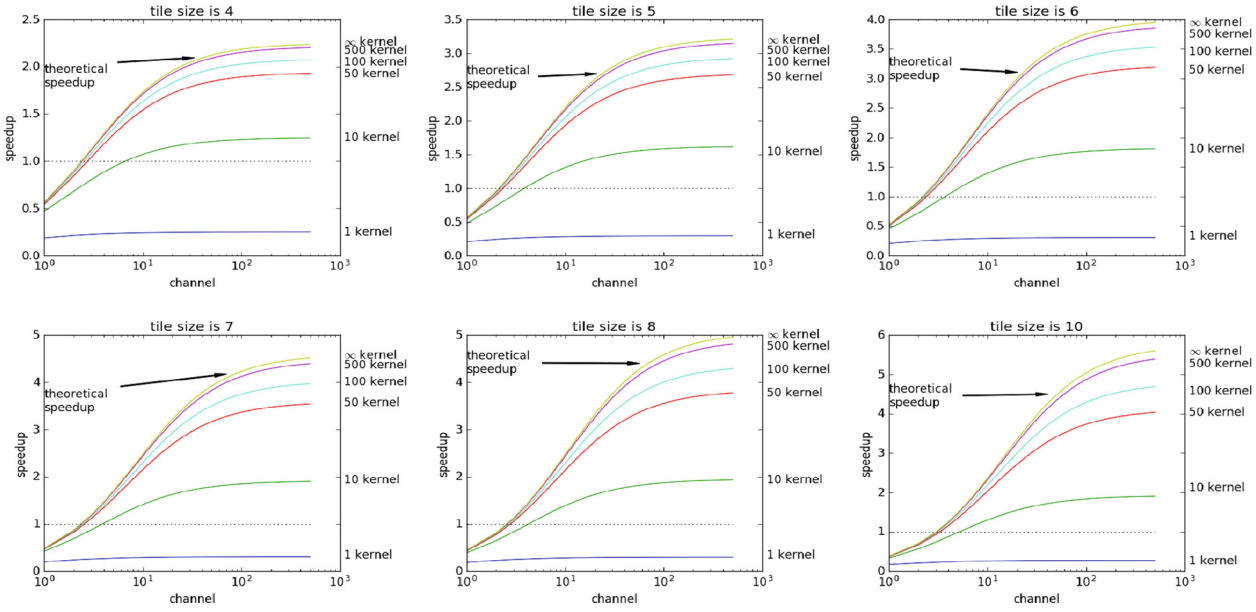

Fig. 2. The speed-up achieved by Winograd convolution for a kernel (3×3) as a function of number layer channels and kernels. Dashed line indicates direct convolution baseline.

5. Optimization methods

5.1. Tile fusion

Through the quantitative analysis in Section 4, compared with 4×4 tile, 6×6 or larger tile can result in a better convolution acceleration effect when the convolution kernel size is fixed at 3×3. However, there is a problem with loss of precision in the Winograd-based fast convolution algorithm. A single-precision floating-point has only 24 bits for the significand, which can dramatically increase the computation error for larger tiles. To benefit from large tiles, we propose a tile-fusion method for CNNs. The method includes structure-based and data-based tile-fusion schemes. The former means that each layer of a CNN performs convolution operations with tiles of different sizes. The latter means that the input data are divided into parts. For each part of the input data a CNN in entered with a tile of a different size. In order to improve the convolution efficiency as much as possible, we will fuse tiles of different sizes within the allowable error range. In Algorithm 1, we calculate the actual parameter values for different slices. In the case of divisible, we use the original parameters. In the case of non-divisible, we will consider zero paddings and recalculate the values of parameters H and W. Finally, the optimal tile size is calculated by the calculation model.

Algorithm 2: Compute optimized tile size for each ConvNet layer

m is the size of the output tile, R is the kernel size.
$T = m + R - 1$ is the size of the input tile.
L is the number of layers.
for $i = 0$ to L **do**
if (w_i-R + 1)modm $\neq 0$ **then**
$w_i = (\lceil w_i - R + 1 \rceil / m) \times m + R - 1$
$h_i = (\lceil h_i - R + 1 \rceil / m) \times m + R - 1$
$T_i = min(F(C_i, N_i, W_i, H_i, K_i, T))$
end for

5.2. Matrix Sparsification

There are two methods for the transform and inverse transform-manual transform and automatic transform. For the manual transform, the matrix multiplication parts in the formula are combined into a one-time calculation, and the repeated calculation part in the process is reduced manually. In this case, the calculation requirement is the least. For the automatic transform, the matrix multiplication part of the formula is divided into two matrix multiplications to facilitate the call of library functions, which have strong scalability; however, the calculation requirement increases correspondingly. In the above-mentioned quantitative analysis model, we found some calculation patterns, which can make better use of the sparsity of matrices A, B, G, and reduce the calculation requirement in the conversion process.

5.3. Batch Size Fusion

In the Winograd algorithm, we can see that the main source of computation is matrix multiplication. For each convolution kernel matrix, multiplications with N input feature map matrices are implemented. Therefore, we can merge these input feature map matrices to compute matrix multiplication with the convolution kernel matrix. This optimization will expand the scale and improve the efficiency of matrix multiplication. On the other hand, it also takes into account the interface of general libraries, such as openBLAS.

6. Experiment

This section presents the results of our experiments to demonstrate the performance of the tile fusion method. We test the tile-fusion method from three aspects:

1. How does the method perform for different convolutional layers?
   The experimental results show that the tile fusion method can compute the optimal performance tile size with acceptable accuracy for all the convolutional layers in the four common models including VGG19 [33], ResNet [34], GoogleNet [35], and SqueezeNet [15].
2. How does the method perform for different model?.
   The experimental results show that for the four mainstream convolutional neural network models, the tile fusion method improves by 1.17 × - 3.57 × compared to the methods in baseline, respectively.
3. What is the error of the tile fusion method comparing with standard convolution methods?. Comparing to the direct convolution method with 64-bit data, the tile fusion method with 32ibt offers an error that is on average lower than E-02, and the error loss is acceptable.

6.1. Experimental Setting

6.1.1. Experimental environment

We benchmark the tile fusion method and baseline methods on the FT1500A platform which contains ARM-v8(16 cores, 1.5 GHz), 32 KB L1 cache, 8 MB L2 cache, 8 MB L3 cache and 16 GB DRAM.

6.1.2. Workload

Winograd-based convolution is widely used in prevalent CNN models, including VGG-19, ResNet, GoogleNet, and SqueezeNet. The models contain a mass of small size filters, which is suited for Winograd-based convolution. For example, 60%of convolution filters in GoogleNet are 3 × 3, 1 × 5, 1 × 7, etc. All the filters in the convolution layers of VGG-19 are 3 × 3. Hence, we select VGG-19, GoogleNet, Resnet, and SqueezeNet to be the baseline model.

6.1.3. Baseline

We organize 4 state-of-the-art implementations as the baseline, which can be divided into two groups: (1) GEMM-based baseline: im2col; (2) Wingrad with fixed tile baseline: 2 × 2 tile(CuDNN), 4 × 4 tile(Arm Compute Library), and 6 × 6 tile(NNPACK).

6.2. For the different convolution layers

We implement the tile fusion method and test it on each convolution layer in different models. Table 1 lists all the convolution layers parameters of the four models above. Take the VGG-19 model as an example, we test the performance of Winograd in different convolution layers with a fixed tile firstly, and the results show a very erratic performance, for instance, the 6 × 6 tiles provide the best performance at the VGG1.1 convolutional layer but poor performance at the VGG 4.2 convolutional layer.

In Fig. 3, the blue bars show that padding problems occur with the tile size, and the gold bar does not generate a padding problem. The dashed lines show the threshold of the tile size. Once the tile size exceeds the threshold, the result of the Winograd-based convolution is not acceptable. For the VGG1.1, 1.2, and 2.1 layers,

Table 1
The parameters of each convolution layers in VGG 19, SqueezeNet, GoogleNet, and ResNet.

Convolution layers	C	H	W	K
Vgg 1.1	3	224	224	64
Vgg 1.2	64	224	224	64
Vgg 2.2	128	112	112	128
Vgg 3.2	256	56	56	256
Vgg 4.2	512	28	28	512
Vgg 5	512	14	14	512
SqueezeNet Fire2	64	55	55	128
SqueezeNet Fire3	64	55	55	256
SqueezeNet Fire5	128	27	27	256
SqueezeNet Fire6	196	27	27	384
SqueezeNet Fire9	256	13	13	512
GoogleNet2	64	56	56	192
GoogleNet3	128	28	28	192
GoogleNet4	144	14	14	288
GoogleNet5	192	7	7	384
ResNet Conv2	64	56	56	64
ResNet Conv3	64	28	28	128
ResNet Conv4	128	14	14	256
ResNet Conv5	256	7	7	512

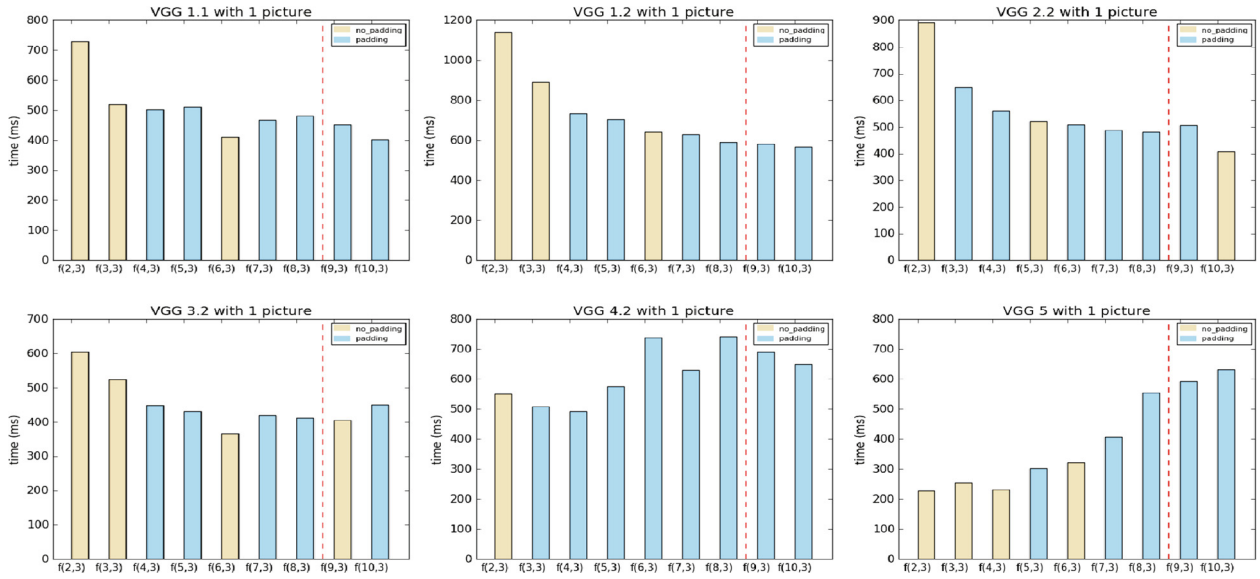

Fig. 3. For each layers of VGG19 E, we test different tile sizes with 1 input feature map.

the cause of the small channel value, the performance curve is U-shaped. In the following layers, such as VGG2.2, because the channel number is greater than 100, the performance curve increases linearly with an increase of the tile value. When the tile size is greater than 8, the gradient of the performance curve slows down. There are some abnormal points on the curve because the width and height of the feature map cannot be divided by the tile size, and additional padding operations are unavoidable. In the VGG4.2 and VGG5 layers, the number of the kernel is 512, the sizes of the feature maps are 28×28, and 14×14, and only one feature map is processed in a batch. Therefore, the number of tiles is very small, which means that the transform cost of 512 convolution filters has not been distributed, and the large tile size increases the total computation overhead.

In Fig. 4, we set the value of batch to 32, and for VGG4.2 and VGG5 layers, the transform cost of the convolutional kernel is shared by the 32 feature maps. Thus, the performance of different tiles meets the description of the computation model. However, when the feature map turns small, the overhead of padding which a large tile introduces covers the benefits it brings. In general, we can observe that the parameters of each convolution layer are different. Running a VGG model with a unique tile, the performance of each layer is instable. In Fig. 5, we show the performance of the tile fusion method on SuqueezeNet, ResNet and GoogleNet models and the results are similar to the VGG model.

6.3. For the different networks

Finally, we test the performance of different tiles in the whole VGG neural network, and the batch size is set to 32. In Fig. 6, we can observe that the overall trend of performance is improved by the increasing tile size. Compared with a fixed tile size, the tile-fusion method solves the problem of unbalanced performance by selecting the optimal tile size for each layer. After its use, the over-

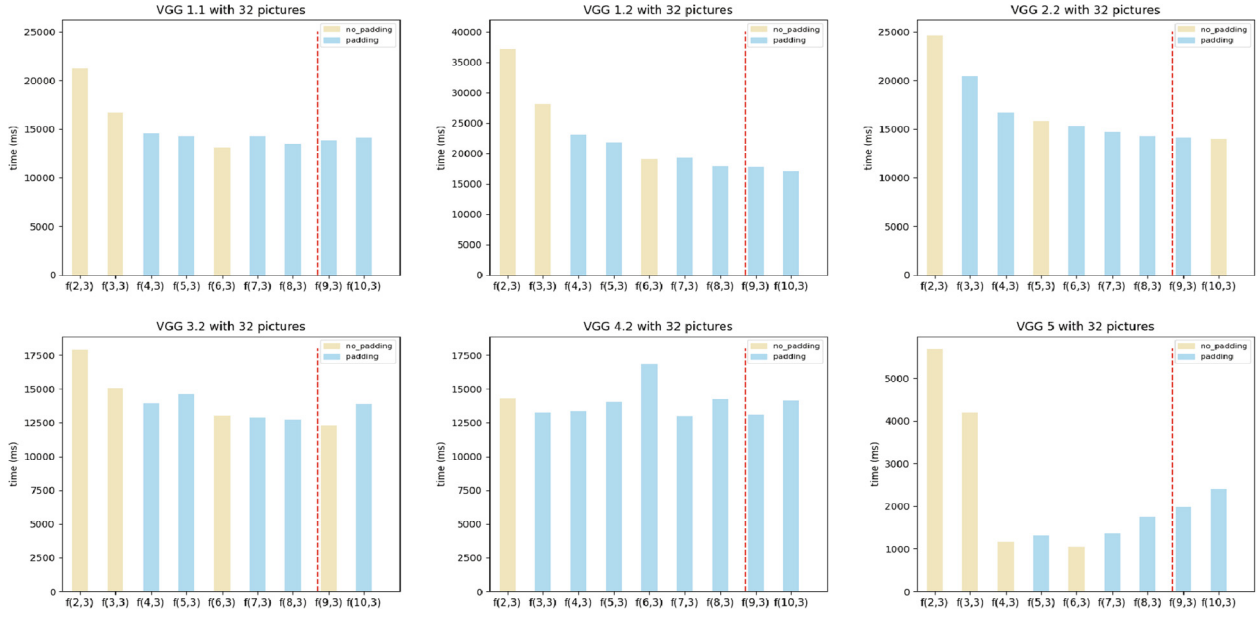

Fig. 4. For each layers of VGG19 E, we test the performance of different tile sizes with 32 input feature maps.

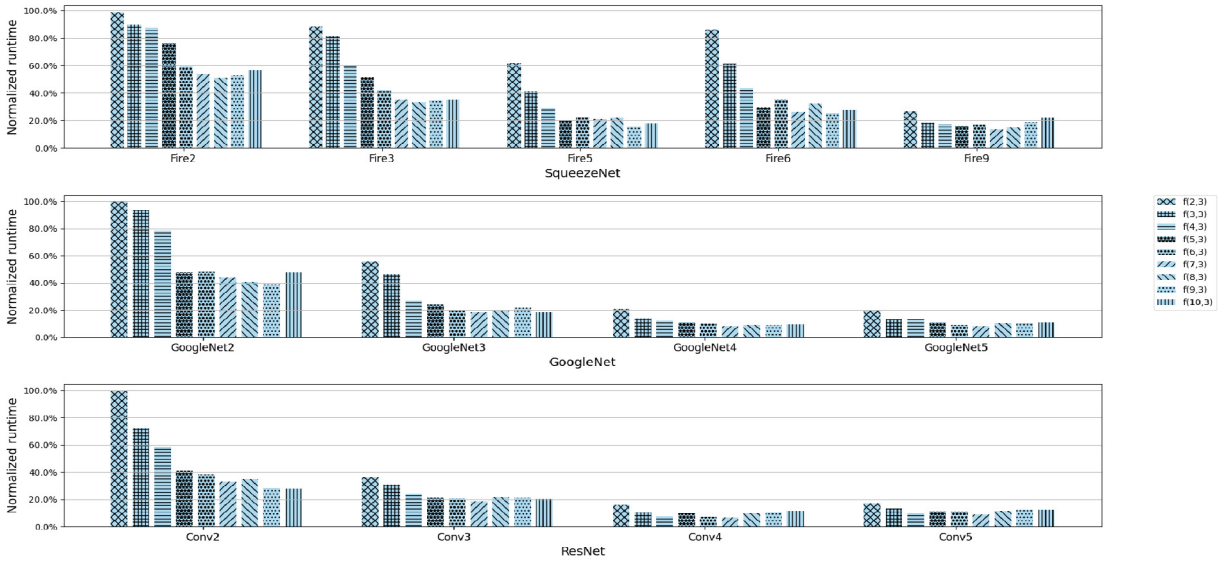

Fig. 5. The performance of different tile sizes with different convolution layers on models SqueezeNet, GoogleNet, and ResNet.

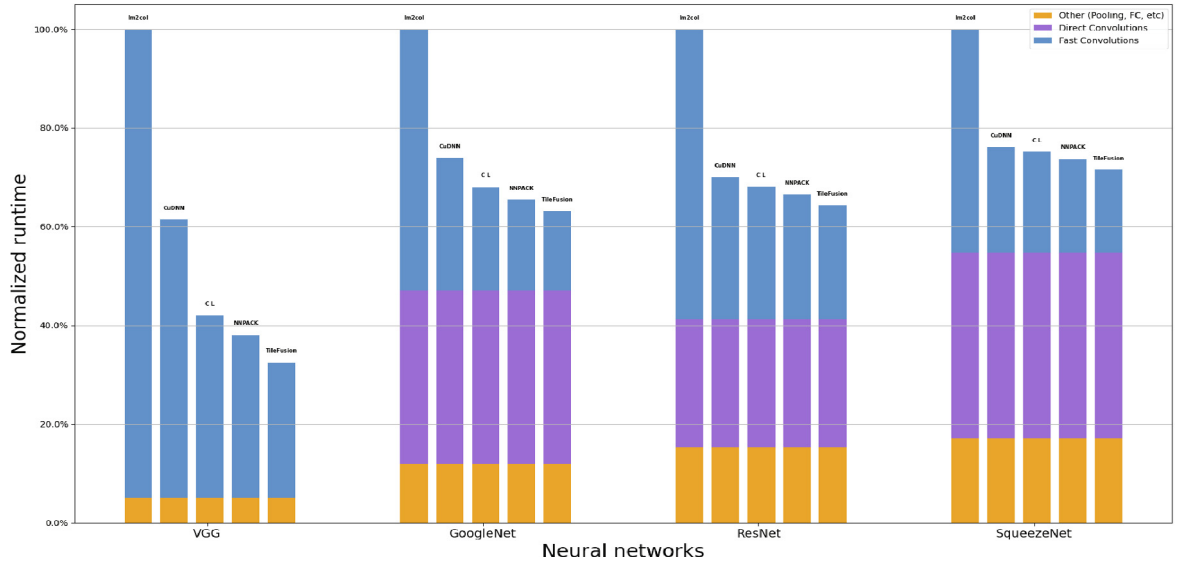

Fig. 6. The performance of different baselines on model VGG, GoogleNet, SqueezeNet, and ResNet.

all performance is improved by approximately 47.2%, 22.1%, and 16.2% respectively, compared with tile size 4, tile size 6, and tile size 8, which are implemented using CuDNN [12], TVM [25], Arm Compute Library [24], and NNPACK [18]. Tests on the models ResNet, GoogleNet, and SqueezeNet presented similar results, as shown in Fig. 6. However, in the three models mentioned above, the tile fusion method does not significantly improve the performance due to the introduction of a 1X1 convolutional kernel which cannot be processed.

6.4. Error

As the tile size increases, higher precision numbers are needed for the Winograd algorithm to ensure the accuracy of the convolution results. Most deep learning algorithms use 32-bit floating-point data, which has a limited number of bits (24) for data. Single-precision floating-point data may result in unacceptable calculation errors for larger tile sizes. Therefore, we measure the calculation error for different tile sizes in order to determine the

Table 2
The calculation error for different tile sizes.

Tile size	4	5	6	7	8	10	12	direct conv
Error max	7.67e-5	7.53e-4	8.45e-4	2.98e-3	8.1e-2	7.31	211228.6	9.81e-5
Error avg	9.94e-8	3.07e-7	3.62e-5	3.67e-5	4.53e-5	6.87e-3	3.98	2.49e-7

acceptable range of tile sizes. Using 32-bit floating-point as a reference, we tested the maximum and average errors for various values of $f(m,r)$ and the error of direct convolution. The direct convolution algorithm using long-double data is used as a ground truth. The input image data and convolution kernel data are uniformly distributed in [0, 1]. Accuracy is more important for training as it affects stability, but has less effect on inference. There is evidence [36,37] that error values under E-02 do not affect the stability of the training and that the inference error may be an order of magnitude higher. Our test results, displayed in Table 2, indicate that when the tile size is less than or equal to 10, the average error is within the acceptable range, and when the tile size is less than or equal to 8, the maximum error and average error are within the acceptable range.

7. Conclusion

Since the Winograd algorithm can significantly reduce the computational complexity of the convolutional layer, it is applied in many popular platforms and machine learning frameworks. Our study is based on the calculation model of the Winograd algorithm, and a quantitative model of the computational complexity was generated. We explored the impact of different tile sizes on CNNs and proposed a series of optimization schemes based on the tile-fusion method. The test results on VGG-19 show that compared with the convolution method that is integrated into the prevalent deep learning framework, the acceleration effect of the CNN is improved by 16% - 47%.

CRediT authorship contribution statement

Zeyu Ji: Conceptualization, Methodology, Software, Investigation, Writing - original draft, Validation. **Xingjun Zhang:** Resources, Funding acquisition, Project administration, Supervision. **Zheng Wei:** Formal analysis. **Jingbo Li:** Investigation. **Jia Wei:** Data curation, Validation.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

We would like to thank the anonymous reviewers, whose insightful comments greatly improved the quality of this paper. The work described in this paper was supported in part by the National Key Research and Development Program of China (2016YFB0200902).

References

[1] Gu. Jiuxiang, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al., Recent advances in convolutional neural networks, Pattern Recogn. 77 (2018) 354–377.
[2] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. arXiv preprint arXiv:1706.02677, 2017. .
[3] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. arXiv preprint arXiv:1708.03888, 2017. .
[4] Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. Image classification at supercomputer scale. arXiv preprint arXiv:1811.06992, 2018. .
[5] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In 11th {*USENIX*} Symposium on Operating Systems Design and Implementation ({*OSDI*} 14), pages 583–598, 2014. .
[6] Frédéric Loulergue, Frédéric Gava, David Billiet, Bulk synchronous parallel ml: modular implementation and performance prediction, in: International Conference on Computational Science, Springer, 2005, pp. 1046–1054.
[7] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. In Advances in neural information processing systems, pages 1223–1231, 2013. .
[8] Eric P. Xing, Qirong Ho, Pengtao Xie, Dai Wei, Strategies and principles of distributed machine learning on big data, Engineering 2 (2) (2016) 179–195.
[9] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Advances in neural information processing systems, pages 693–701, 2011. .
[10] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Yoshua Bengio and Yann LeCun, editors, 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2–4, 2016, Conference Track Proceedings, 2016. .
[11] Xingyu Liu, Jeff Pool, Song Han, and William J. Dally. Efficient sparse-winograd convolutional neural networks. In 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net, 2018. .
[12] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, Yoshua Bengio, Quantized neural networks: Training neural networks with low precision weights and activations, J. Mach. Learn. Res. 18 (1) (2017) 6869–6898.
[13] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. arXiv preprint arXiv:1405.3866, 2014. .
[14] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017. .
[15] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. arXiv preprint arXiv:1602.07360, 2016. .
[16] Jiarui Fang, Haohuan Fu, Wenlai Zhao, Bingwei Chen, Weijie Zheng, and Guangwen Yang. swdnn: A library for accelerating deep learning applications on sunway taihulight. In 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pages 615–624. IEEE, 2017. .
[17] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia, pages 675–678, 2014 .
[18] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. arXiv preprint arXiv:1410.0759, 2014 .
[19] Minsik Cho and Daniel Brand. Mec: memory-efficient convolution for deep neural network. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 815–824, 2017. .
[20] Michaël Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. In Yoshua Bengio and Yann LeCun, editors, 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14–16, 2014, Conference Track Proceedings, 2014. .
[21] Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A gpu performance evaluation. arXiv preprint arXiv:1412.7580, 2014. .
[22] Andrew Lavin, Scott Gray, Fast algorithms for convolutional neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4013–4021.
[23] Shmuel Winograd. Arithmetic complexity of computations, volume 33. Siam, 1980. .
[24] Arm compute library. [EB/OL]. https://github.com/ARM-software/ComputeLibrary. .
[25] Acceleration package for neural networks on multi-core cpus: Maratyszcza. [EB/OL]. https://github.com/Maratyszcza/NNPACK. .
[26] David Budden, Alexander Matveev, Shibani Santurkar, Shraman Ray Chaudhuri, and Nir Shavit. Deep tensor convolution on multicores, in: International Conference on Machine Learning, 2017, pp. 615–624.
[27] Partha Maji, Andrew Mundy, Ganesh Dasika, Jesse Beu, Matthew Mattina, and Robert Mullins. Efficient winograd or cook-toom convolution kernel implementation on widely used mobile cpus. In 2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2), pages 1–5. IEEE, 2019. .
[28] Zhen Jia, Aleksandar Zlateski, Fredo Durand, Kai Li, Optimizing n-dimensional, winograd-based convolution for manycore cpus, in: Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2018, pp. 109–123.
[29] Junzhong Shen, You Huang, Zelong Wang, Yuran Qiao, Mei Wen, and Chunyuan Zhang. Towards a uniform template-based architecture for accelerating 2d and 3d cnns on fpga. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pages 97–106, 2018. .
[30] Feng Shi, Haochen Li, Yuhe Gao, Benjamin Kuschner, Song-Chun Zhu, Sparse winograd convolutional neural networks on small-scale systolic arrays, in: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2019, p. 118.

[31] Lanmin Zheng and Tianqi Chen. Optimizing deep learning workloads on arm gpu with tvm. In Proceedings of the 1st on Reproducible Quality-Efficient Systems Tournament on Co-designing Pareto-efficient Deep Learning, page 1. 2018. .
[32] Mkl. [EB/OL]. https://github.com/oneapi-src/oneDNN. .
[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014. .
[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
[35] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1–9, 2015. .
[36] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. arXiv preprint arXiv:1412.7024, 2014. .
[37] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, Pritish Narayanan, Deep learning with limited numerical precision, in: International Conference on Machine Learning, 2015, pp. 1737–1746.

Zeyu Ji received the B.S.degree from the School of Information Engineering, Zhengzhou University, Zhengzhou, China, in 2008, and the M.S. degree from the Polytechnic University of Tours, Tours, France. He is currently a PhD candidate with Xi'an Jiaotong University, Xi'an, China.. His main research interests include computer architecture, high performance computing, and deep learning.

Xingjun Zhang received his Ph.D degree in Computer Architecture from Xi'an Jiaotong University, China, in 2003. From Jan. 2004 to Dec. 2005, he was Postdoctoral Fellow at the Computer School of Beihang University, China. From Feb. 2006 to Jan. 2009, he was Research Fellow in the Department of Electronic Engineering of Aston University, United Kingdom. He is now a Full Professor and the Dean of the School of Computer Science & Technology, Xi'an Jiaotong University. His research interests include high performance computing, big data storage system and machine learning acceleration.

Jia Wei received the B.E.degree from the School of Information science and technology, NorthWest University, Xi'an, China, in 2019. He is currently a PhD candidate with the Computer science and technology School, Xi'an Jiaotong University. His research interests include computer architecture, high performance computing, and deep learning.

Jingbo Li is currently pursuing a Ph.D. degree with Xi'an Jiaotong University, Xi'an, China. His main research interests include computer architecture, job scheduling, and high performance computing.

Zheng Wei received the B.S. and M.S degrees from the school of Communication Engineering from Xidian University, Xi'an, China, in 2013 and 2016, respectively. He is currently pursuing a Ph.D. degree with Xi'an Jiaotong University, Xi'an, China. His research interests include computer architecture, deep compression algorithms, and hardware accelerators for deep learning.