Optimizing Convolution Neural Network on the TI C6678 multi-core DSP

Guozhao Zeng^{1, a}, Xiao Hu¹ and Yueyue Chen¹

¹College of Computer, National University of Defence Technology, Changsha, Hunan Province, China

Abstract: Convolutional Neural Networks (CNNs) have become the most advanced algorithms for deep learning. They are widely used in image processing, object detection and automatic translation. As the demand for CNNs continues to increase, the platforms on which they are deployed continue to expand. As an excellent low-power, high-performance, embedded solution, Digital Signal Processor (DSP) is used frequently in many key areas. This paper attempts to deploy the CNN to Texas Instruments (TI)'s TMS320C6678 multi-core DSP and optimize the main operations (convolution) to accommodate the DSP structure. The efficiency of the improved convolution operation has increased by tens of times.

1 Introduction

In recent years, deep learning has emerged in the rapid development of computer hardware, and has become a popular technology. The Convolutional Neural Network (CNN) has achieved remarkable results in the field of computer vision. Because CNNs use a large number of convolutional layers, the computational complexity of the network is increased, and a huge workload is brought about. At present, we often use GPUs to accelerate the training and inference processes of CNNs. As technology continues to develop and application needs, the overhead of GPU area and power consumption becomes unbearable, so deploying CNNs on mobile and embedded platforms is beginning, such as FPGAs and ASICs. Digital signal processors (DSPs) are known for their low power consumption, high computing power, and ease of programming. A large number of MAC units have a natural advantage for convolution operations. This paper uses Texas Instruments (TI)'s TMS320C6678 for the deployment and optimization of CNNs.

The TMS320C6678 is a multi-core DSP based on TI's Keystone architecture. It has 8 C66x cores and supports 1GHz or 1.25GHz frequency, up to 320GMAC or 160GFLOPs, and consumes only 10W. The TMS320C6678 combines high performance with low power consumption and is the first choice for High-performance Embedded Computing (HPEC).

As shown in Fig. 1, the C6678 has eight C66x CorePacs based on Very Long Instruction Words (VLIW). Each core has 32KB of L1P, L1D and 512KB of L2 SRAM memory, and multi-core shared memory MSM SRAM with a capacity of 4MB. The processor also has some external resources that are shared with multiple cores such as multi-core navigators, network coprocessors, packet accelerators, and semaphores.

^a Corresponding author: zgzh1993@hotmail.com

Figure 1. C6678 Architecture.

As shown in Fig. 2, TI's latest C66x core consists of two symmetrical parts (A, B), each with four functional units (.D, .L, .S, .M), which can execute one instruction per cycle. Among them, the .M unit is responsible for the multiplication instruction, the .S unit and the .L unit are responsible for the execution of the arithmetic, logic and branch programs, and the .D unit is mainly responsible for reading the data. A .M unit contains 16 16-bit multipliers, so the C66x core can perform four single-precision multiply (32bit) per cycle, making it the most performing floating-point DSP on the market. The perfect integration of multiple C66x DSP cores creates a multi-core System-on-Chip (SoC) device with superior performance.

Figure 2. C66x multiplication unit. 16 16-bit multipliers included.

2 Convolution method

Convolution operation is the main operation of CNNs, and it is also the most time-consuming part. The convolution is calculated as Equation 1, where kw is the width of convolution kernel, and kh is the height of convolution kernel. sw is the step of the sliding window in the horizontal direction, and sh is the step of the sliding window in the vertical direction. xn represents the number of channels of the input image, $o(n,i,j)$ represents the output value of (i,j) in the n th output image, and $x(m,i,j)$ represents the m th input image (i,j) , $w(n,m,u,v)$ represents the value of weight, n corresponds to the channel of the output image, m corresponds to the channel of the input image, and u and v represent the coordinate of a single convolution kernel.

$$o(n, i, j) = \sum_{m=0}^{xn-1} \left(\sum_{v=0}^{kh-1} \left(\sum_{u=0}^{kw-1} w(n, m, u, v) x(m, u + i * sw, v + j * sh) \right) \right) \quad (1)$$

In order to improve the efficiency of the operation, it is often tried to train and derive the CNN using different convolution methods. Common convolution implementation methods include sliding windows, GEMM, Winograd algorithm, and FFT algorithm.

2.1 Sliding windows

The sliding windows method is to perform the operation step by step according to the Equation 1, and to multiply and accumulate the input data and the convolution kernel by means of loop nesting and data index. The main implementation process is as follows:

```

for(n=0;n<N;n++) { // channel of output
    for(i=0;i<R;j++) { // height of output
        for(j=0;j<C;i++) { // width of output
            for(m=0;m<M;m++) { // channel of input
                for(u=0;u<Kh;u++) { // height of kernel
                    for(v=0;v<Kw;v++) { // width of kernel
                        output[n][i][j] += weights[n][m][u][v]*input[n][ sw*i+u][ sh*j+v];
                    } } } } } }

```

2.2 GEMM

GEMM is an effective convolution method used by mainstream deep learning frameworks such as Caffe, MXNet, etc. This method use *im2col* to transforms the entire convolution process into a matrix multiplication (GEMM in BLAS), and GEMM is extremely optimized in various BLAS libraries. As shown in Fig. 3, this convolution method rearranges the input data in the order of convolution kernel data to ensure that the two are in one-to-one correspondence, and the addresses are consecutive. Although the matrix multiplication takes time and space to tile input data, it guarantees the continuity of the address, which is still more beneficial to the hardware implementation and shortens the time it takes for the convolution operation.

2.3 Winograd algorithm

The Winograd algorithm is a fast convolution algorithm that derives from Winograd's minimum filtering algorithm. The

Figure 3. The convolution step of Caffe, using im2col firstly. After that, matrix multiplication of weights and inputs.

Winograd algorithm reduces the number of multiplications with a series of transformations on the input data and the convolution kernel. Simply put, more addition calculations are used to reduce the multiplication. For example, a one-dimensional convolution with an output size of m and a convolution kernel size of r , a normal convolution operation requires $m \times r$ multiplication, and a Winograd algorithm requires $(m + r - 1)$ multiplication. The Winograd algorithm can be a good acceleration for the platforms where the multiply computing clock period is much larger than the addition calculation clock period.

2.4 FFT algorithm

Fourier transform and fast Fourier transform are the calculation methods often used in classical image processing algorithms, but they are not usually used in CNN because the convolution kernel size of CNN is usually small, such as 1×1 , 3×3 , etc. In this case, the time overhead of the FFT is even larger. Therefore, unless CNN uses a relatively large convolution kernel, the time overhead of the FFT can be hidden.

3 Deployment on DSP

3.1 CNN architecture

The CNN was a biophysical model for the recognition of two-dimensional shapes, which was originally inspired by the neural mechanism of the visual system. The CNN can be regarded as a special multi-layer perceptron or feedforward neural network. In the case of translation, it is highly invariant and has certain invariance in the case of scaling and tilting. In addition, the CNN also has the characteristics of local connection and weight sharing, which reduces the amount of calculation and the number of parameters. CNNs excel in many fields, especially in image recognition.

The CNN mainly includes convolutional layers, pooling layers, and fully connected layers, and its general structure is as shown in Fig. 4. Among them, the convolution layers are the main structure of the whole network. In convolution layers, several filters are applied to extract different types of features from the input data. The role of the pooling layer is to downsample the input feature map and reduce the size of the input data. The most common pooling method is the MaxPooling, which takes the maximum value of each part for output. After extracting the feature, the CNN will typically connect one or more fully connected layers at the end for classification.

Although the calculation of the fully connected layer is far behind the convolutional layer, the parameter ratio is the largest among the entire network parameters.

3.2 Deployment

We are committed to deploying a common CNN framework on the DSP. The mainstream deep learning frameworks now include Caffe, Tensorflow, MXNet, etc., but these frameworks rely on a number of different operating system-specific libraries and are not suitable for DSP embedded systems.

We write our framework in C. Since only forward derivation is required, we only need to implement the calculation of the Convolution Layer, MaxPooling Layer, the Fully-Connected Layer, and the data transfer. In order to maintain the accuracy of the data, all data uses a single-precision floating point type. The input data and convolution kernel weight values are stored in the DDR, and the required data is preloaded into L1 and L2 during the calculation.

In the implementation of convolution, we have measured several convolution methods. At the same time, we optimized the matrix multiplication for the characteristics of the C66x DSP. As shown in Table I, we measured three convolution methods: sliding windows, GEMM, and optimized matrix multiplication with AlexNet on the TMS320C6678. It is not difficult to see that our optimized matrix multiplication is much more efficient than using GEMM in Caffe directly, and even improves the efficiency by hundreds of times.

When implementing matrix multiplication, we use the inline function `_ftod` to combine adjacent data, divide the whole matrix into 2×2 small matrices, and use DMPYSP, DADSP 2-way SIMD single-precision addition and single-precision multiplication instructions to block matrix multiply and accumulate operations. Through loop expansion and scheduling, 8 times of multiplying and adding operations per loop, making full use of the two functional units of DSP to improve parallelism, the actual efficiency can reach $\sim 2\text{GFLOP} / \text{s}$.

```
sum0 = _daddsp(sum, _dmpysp(x0, y0);
sum1 = _daddsp(sum, _dmpysp(x1, y1);
sum2 = _daddsp(sum, _dmpysp(x2, y2);
sum3 = _daddsp(sum, _dmpysp(x3, y3);
sum4 = _daddsp(sum, _dmpysp(x4, y4);
sum5 = _daddsp(sum, _dmpysp(x5, y5);
sum6 = _daddsp(sum, _dmpysp(x6, y6);
sum7 = _daddsp(sum, _dmpysp(x7, y7)
```

Figure 4. The CNN architecture is mainly composed of convolutional layers, pooled layers and fully connected layers.

Table 1. Comparison of 3 convolution methods

Input Size	Output Size	Kernel Sizes	Time		
			Sliding Windows	GEMM	Optimized Matrix
227×227×3	55×55×96	11×11	1.67s	1.74s	0.11s
27×27×96	27×27×256	5×5	9.15s	7.26s	1.78s
13×13×256	13×13×384	3×3	3.68s	2.45s	0.72s
13×13×384	13×13×384	3×3	5.62s	3.67s	1.25s
13×13×384	13×13×256	3×3	3.75s	2.46s	0.80s

The pooling function replaces the output of the network at that location using the overall statistical characteristics of the adjacent outputs at a location. The maxpooling is to take the maximum output in the adjacent rectangular area, so only logical comparison is needed, and a lot of calculations are not needed. The fully connected layer can be thought of as a global convolution with the same convolution kernel size and input feature map size. In the implementation, the method of referring to convolution is carried out.

3.3 Parallel

After completing the deployment, we begin parallel optimization. The TMS320C6678 has 8 cores. It is obviously unscientific to perform 8-core parallel operations on all tasks. Instead, it may cause communication time to be greater than the operation time. Through *profile* analysis, the time of project is mainly used for convolution calculations, so we only perform parallel acceleration for convolution.

There are two main types of parallel processing models: master-slave model and data stream model. The model structure is shown in Fig. 5 and Fig 6. The master-slave model depicts a model that controls concentration and execution distribution, while the data flow model represents distributed control and execution.

Figure 5. Master-Slave model

Figure 6. Data stream model

We already know that the most time-consuming layer of CNN is convolutional layer, and the whole network is streamlined, dependent on the front, not independent, so it is more suitable for the master-slave parallel model. In the aspect of multi-core communication, IPC is commonly used for communication, but we find that the time of interruption event is large, especially for small convolutional layers. Therefore, we use semaphore query to communicate. The specific implementation is as follows:

- The main core occupies a semaphore A;
- From the core query to the time when the semaphore is occupied, the loop starts to perform the operation, and the core also occupies one semaphore;
- The operation is completed, and all semaphores are released from the core;
- The main core queries the operation after the release of the core semaphore and continues execution.

The CSL library gives us a semaphore synchronization function:

```

CSL_semAcquireDirect( ); // Synchronization semaphore
//Convolution operation;
CSL_semReleaseSemaphore(); //Free semaphore

```

We test the speed of convolution after parallel acceleration, as shown in Table II, which is several times faster than before, especially in the case of a large number of convolution operations.

Table 2. Speedup of multi-core parallel.

FLOPs	Time of Single-Core	Time of Multi-Core
10M	0.01s	0.004s
100M	0.06s	0.02s
1G	0.61s	0.20s
10G	5.37s	1.07s
100G	55.52s	8.95s

4 Conclusion

This paper introduces the basic structure of CNNs and deploys them to TI's TMS320C6678 platform. The most time-consuming layer of CNN on the DSP is the convolutional layer. Therefore, we analyze several commonly-used convolution methods, and optimize and parallelize convolution calculations according to the characteristics of TMS320C6678, which shortens the computation time 30 to 60 times.

The data storage in this paper uses single-precision floating-point numbers. The next step is to optimize the fixed-point number. Based on this, the pooling layer can be further optimized. At the same time, many CNNs have a large number of weights, and it is worthwhile to study the compression and tailoring of weights when transplanting to embedded platforms.

References

1. Jagannathan S, Mody M, Mathew M. Optimizing convolutional neural network on DSP[C]// IEEE International Conference on Consumer Electronics. IEEE, 2016:371-372.
2. Ramesh B, Bhardwaj A, Richardson J, et al. Optimization and evaluation of image- and signal-processing kernels on the TI C6678 multi-core DSP[C]// High PERFORMANCE Extreme Computing Conference. IEEE, 2015:1-6.
3. Jia, Yangqing, Shelhamer, et al. Caffe: Convolutional Architecture for Fast Feature Embedding[J]. 2014:675-678.
4. S. Mu, C. Wang, M. Liu, D. Li, M. Zhu, X. Chen, X. Xie, and Y. Deng, "Evaluating the potential of graphics processors for high performance embedded computing," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1–6.
5. TMS320C66x DSP Core Pac User Guide. [Online]. Available:
<http://www.ti.com/lit/ug/sprugw0c/sprugw0c.pdf>
6. TMS320C6678 multicore fixed and floating-point digital signal processor, data manual. [Online]. Available:
<http://www.ti.com/lit/ds/sprs691c/sprs691c.pdf>
7. TMS320C66x DSP CPU and Instruction Set Reference Guide. [Online]. Available:
<http://www.ti.com/lit/ug/sprugh7/sprugh7.pdf>