electronics

MDPI

Article

Efficient Distributed Mapping-Based Computation for Convolutional Neural Networks in Multi-Core Embedded Parallel Environment

Long Jia [1], Gang Li [2], Meili Lu [3], Xile Wei [4] and Guosheng Yi [4,*]

[1] School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China; truelongvip@sina.com
[2] Beijing Aerospace Automatic Control Institute, Beijing 100854, China; lig23@163.com
[3] School of Information Technology Engineering, Tianjin University of Technology and Education, Tianjin 300222, China; meililu@tute.edu.cn
[4] Tianjin Key Laboratory of Process Measurement and Control, School of Electrical and Information Engineering, Tianjin University, Tianjin 300072, China; xilewei@tju.edu.cn
* Correspondence: guoshengyi@tju.edu.cn

Abstract: Embedded systems are the best solution to achieve high-performance edge terminal computing tasks. With the rapid increase in the amount of data generated by edge devices, it is imperative to implement intelligent algorithms with large amounts of data and computation on embedded terminal systems. In this paper, a novel multi-core ARM-based embedded hardware platform with a three-dimensional mesh structure was first established to support the decentralized algorithms. To deploy deep convolutional neural networks (CNNs) in this embedded parallel environment, a distributed mapping mechanism was proposed to efficiently decentralize computation tasks in the form of a multi-branch assembly line. In addition, a dimensionality reduction initialization method was also utilized to successfully resolve the conflict between the storage requirement of computation tasks and the limited physical memories. LeNet-5 networks with different sizes were optimized and implemented in the embedded platform to verify the performance of our proposed strategies. The results showed that memory usage can be controlled within the usable range through dimensionality reduction. The down-sampling layer as the base point of the mapping for the inter-layer segmentation achieved the optimal operation in lateral dispersion with a reduction of around 10% in the running time compared with the other layers. Further, the computing speed for a network with an input size of 105 × 105 in the multi-core parallel environment is nearly 20 times faster than that in a single-core system. This paper provided a feasible strategy for edge deployments of artificial intelligent algorithms on multi-core embedded devices.

Keywords: edge computing; convolutional neural network; parallel computing; embedded platform; distributed mapping

check for updates

Citation: Jia, L.; Li, G.; Lu, M.; Wei, X.; Yi, G. Efficient Distributed Mapping-Based Computation for Convolutional Neural Networks in Multi-Core Embedded Parallel Environment. *Electronics* **2023**, *12*, 3747. https://doi.org/10.3390/electronics12183747

Academic Editors: Claus Pahl and Carlo Mastroianni

Received: 15 June 2023
Revised: 17 August 2023
Accepted: 29 August 2023
Published: 5 September 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Although traditional cloud computing provides a platform for big data processing [1], with the rapid development of the big data area, cloud computing is not as efficient as it used to be in processing large amounts of data [2]. We need a technology that brings computing, resources and storage closer to edge devices. Edge computing is a new computing model [3]. This model deploys computing and storage resources at the edge of the network, closer to mobile devices or sensors. Edge computing transfers data, storage, etc., to the edge, and meets real-time task requirements with lower power consumption [4]. Therefore, the large-scale computing tasks are also transferred to the edge, including convolutional neural networks [5,6]. CNN is one of the most important deep neural networks, and plays an important role in tasks related to computer vision. It has a wide range of applications

in image classification [7,8], image semantic segmentation [9,10], speech recognition [11], target detection [12,13] and target tracking [14,15]. Deep learning algorithms based on convolutional neural networks involve many floating point operations. CNN models mostly run in the environment of CPU [16] and GPU [17,18]. Although GPU can achieve real-time processing, its expensive cost and high power consumption make it difficult to satisfy the application requirements of edge computing scenarios. ARM architecture series microcontrollers are widely used in the industrial field due to their high performance, low cost and abundant software development support. The performance improvement of CNN is mainly driven by deeper and wider networks with increased parameters and operations (e.g., multiply-and-accumulate, MAC), which usually slow down their execution, especially on mobile devices. This is even more important for these mobile devices. Because single core ARM chips do not have the ability to perform parallel computing, their computing speed is far inferior to that of CPUs and GPUs. This motivates the design of compact models with reduced overhead while maintaining accuracy as much as possible. Therefore, it is of great significance to develop a low-cost, low-power, easy-to-develop and parallel computing embedded hardware platform for edge computing. In a prior survey [19], the author summarized and reviewed the current development of machine learning in embedded microprocessors, compared the performance of various neural network algorithms on embedded platforms, and proposed a concept of model compression. However, no specific suggestions for platform construction were provided for specific microprocessors. In addition to improving the computing power of a single microprocessor, improving the system structure and improving the communication and data transmission efficiency of the system can also significantly enhance the computing power of embedded platforms. At the same time, this can simplify and compress the CNN algorithm to make it suitable for distributed operations and small-scale operations of ARM processors. In previous studies, there were still those that proposed decomposing the convolutional kernel of CNN, actively using smaller or asymmetric kernels to simplify the convolutional layer of CNN, reducing the computational complexity from O (n^2) to O (2n), saving a lot of computational overhead [20]. In [21], the authors discuss several techniques for reducing the computational complexity of CNNs; one approach is network pruning and another technique is quantization, which can make these models more feasible for deployment on devices with limited resources, such as mobile phones or embedded systems. There have also been studies on optimizing CNN and leNet-5 models for embedded platforms, such as FPGA, implementing a mixed stream transmission architecture for LeNeT-5, and accelerating single engine computing for LeNeT-5 [22]. These studies provide ideas for simplifying CNN algorithms and making them more suitable for small-scale embedded computing platforms. The above studies provide ideas for innovative simplification and improvement of CNN algorithms to make them more suitable for small-scale embedded computing platforms.

In this work, we try to build an ARM-based CNN parallel computing hardware platform to provide the possibility of embedded edge computing for neural networks. We propose solutions to several key issues in the implementation process. We try to propose a CNN distributed computing method and build an ARM-based CNN parallel computing hardware platform to provide the possibility of embedded edge computing for neural networks. We propose solutions to several key issues in the implementation process. Specifically, the main contributions of this work include the following.

The front-end image acquisition and processing board is designed based on the hardware architecture of the CNN computing board for accomplishing image classification, target tracking and other front-end image acquisition tasks, as well as the distributed parallel processing of images. The front-end image acquisition and processing board and the bottom computing board designed in the laboratory constitute the CNN parallel computing hardware platform. We adopt the idea of time division multiplexing to realize multi-chip shared external static random access memory (SRAM) access. The routing unit controls and schedules the computing unit to access external storage in an orderly loop. Resource

sharing and data integration are realized, and conflicts are avoided in the access process. At the same time, the routing unit is used to realize the board communication. In order to solve the problem of limited resources on the chip, a method of dimensionality reduction initialization is designed. In order to ensure the real-time calculation of the model under the selected chip operating frequency, according to the hierarchical structure of the CNN and the characteristics of the sliding window operation in the convolution process, the distribution mapping mechanism of the neural network is designed. In this way, the neural network tasks are distributed to multiple computing units to achieve parallel computing. The most time-consuming exponential function calculation in the nonlinear calculation process is optimized. At the cost of reducing a small part of the calculation accuracy, the exponential function calculation speed is greatly improved. We verify the effectiveness of the above method in large-scale real-time implementation of CNNs through experiments. Due to the convenience of ARM core software development, the hardware platform design is versatile and easy to implement with other network models.

The rest of this paper is organized as follows. We introduce, in Section 2, the structure design of the hardware platform, including the terminal acquisition and processing board. The external resource sharing mechanism, inter-chip communication and board communication mechanism for this platform are proposed. Section 3 presents the dimensionality reduction initialization and the distribution mapping mechanism of the convolutional neural network model suitable for this platform. The nonlinear calculation process in the network is optimized. After that, we present experimental validation and result analysis in Section 4. Experiments are designed to prove the effectiveness of the above method. The resource consumption and power consumption of the platform are analyzed. Finally, the paper is concluded in Section 5.

2. Embedded Parallel Computing Platform

2.1. Multi-Core Embedded Parallel Platform

In this work, we consider a single ARM as the smallest compute unit (CU) and propose a multi-core local extension structure known as the basic extension module (BEM) based on this CU. Within the BEM, multiple CUs are employed to implement parallel pipelined forward inference in part of the middle layer. The shared read-only memory (ROM) and static random-access memory (SRAM) connect all CUs, storing both model parameters and computational data. The calculated results from each CU are synchronously integrated for subsequent BEM use. We employ a routing unit (RU) that is fully connected to the CUs for cyclic control. Furthermore, inter-layer expansion is achieved through the connection of RUs in different BEMs. The overall architecture is depicted in Figure 1, where BEMs of the same layer perform parallel computing, and BEMs of different layers execute the inter-layer pipeline.

The STM32F407 series chips with ARM structure are selected as CU and RU. The STM32F407 microcontroller comes with 1 MB (megabyte) of built-in ROM, which is divided into two areas: Main Flash memory and System memory. The Main Flash memory, with a size of 1 MB, is used to store application code, while the System memory, with a size of 64 KB (kilobytes), is used to store Bootloader code, EEPROM emulator, and other system-related data. The access speed of Flash memory in the STM32F407 microcontroller depends on various factors such as access mode and operating frequency, and can reach a maximum access speed of 30 MHz. Additionally, the STM32F407 features an Adaptive Real-Time (ART) accelerator, which utilizes prefetching and caching techniques to speed up Flash memory access and improve code execution efficiency. According to the above architecture, a multicore embedded parallel computing platform (MEPP) is designed. The MEPP includes three BEMs and one Front End Acquisition Module (FAM). A BEM consists of eight CUs and one RU. FAM includes four CUs, one RU and a camera module, mainly for image acquisition. In addition, both BEM and FAM include GPIO, USART, USB and other expansion interfaces. We validate the above architecture by implementing CNNs

on computational tasks. Figure 2 shows the actual platform. MEPP can be extended to arbitrary mesh and tree topologies.

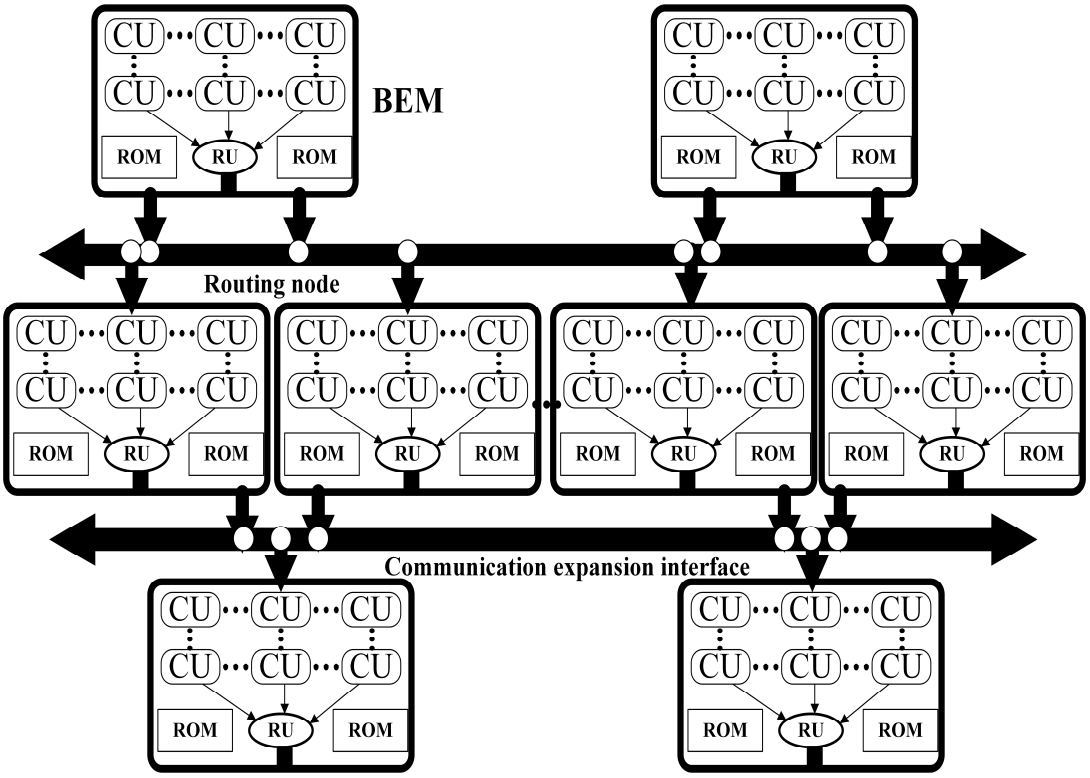

Figure 1. Modular and hierarchical off-chip multi-core computing architecture.

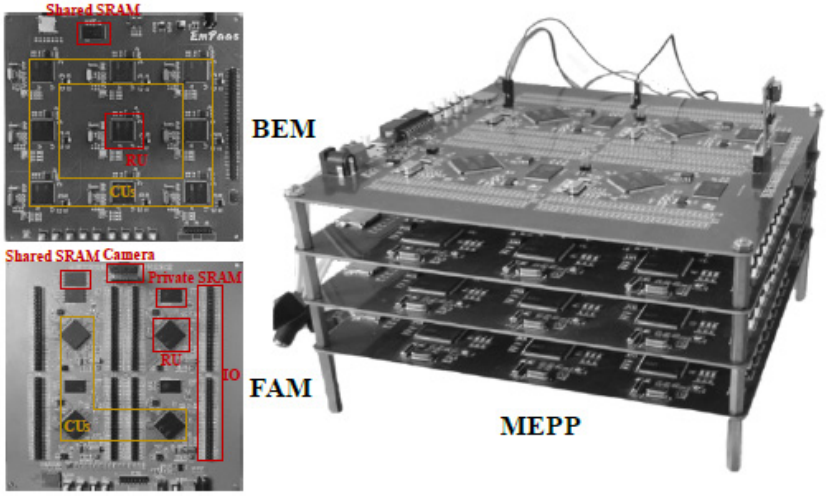

Figure 2. The physical map of BEM, FAM and MEPP.

2.2. CU Shared Resource Access and Communication Strategy

To realize the distributed parallel computing of the convolutional neural network, the shared resource access and communication part is very important. Both the neural network computing board and the front-end acquisition and processing board are equipped with external shared storage resources, including two resource blocks, external RAM and external ROM. Because the speed of shared external RAM reads and writes is fast, it is used to realize shared data access, shared data storage and communication tasks among multiple CUs. The shared external ROM is mainly used to store the model network structure and

parameters during the initialization process. Due to the large storage space of the ROM, historical data, such as the collected image data and the calculation results of each layer of the network model, can be stored. It is helpful for the model training of the image data collected in the early stage and the analysis of the calculation process of the network model. In order to avoid conflicts caused by multiple CUs' access to shared external resources, we design a mechanism to prevent access conflicts for shared external resources suitable for this platform. Before accessing external shared resources, the CU must send an application to the RU. The RU checks whether it is free. If it is idle, the RU controls the CU to visit sequentially and cyclically. We set an access flag to reject other CUs' access applications during the visit. It blocks CUs trying to access the shared storage until the shared storage is released. The next CU visits in turn.

The implementation process of the shared resource access and storage mechanism is shown in Figure 3a. CUs are connected to shared external SRAM through the FSMC interface in the chip. The FSMC interface includes a 20-bit wide address bus and a 16-bit wide data bus. We design a group of fully connected GPIO ports named IOCOM as external SRAM access signs. Through its read and write operations, the external SRAM status can be set and acquired. IOCOM is a low potential which means that it is idle, and a high potential means that it is occupied. According to the nature of parallel computing, CUs are required to complete the same-layer computing synchronously. If the external SRAM is free, the next CU is allowed to access according to the access sequence of the CUs. A group of GPIOs is connected between each CU and RU to realize the handshake operation. The CU occupying the external storage sets IOCOM to indicate that the external SRAM is occupied. The FSMC interface is multiplexed into the address and data bus of the external SRAM. The CU prepares to access data or store data. After reading and writing, release the FSMC interface and set the IOCOM bit to zero. When the CU does not generate a demand for accessing external SRAM, its FSMC interface must be released and configured to stop working. Figure 3b describes this process.

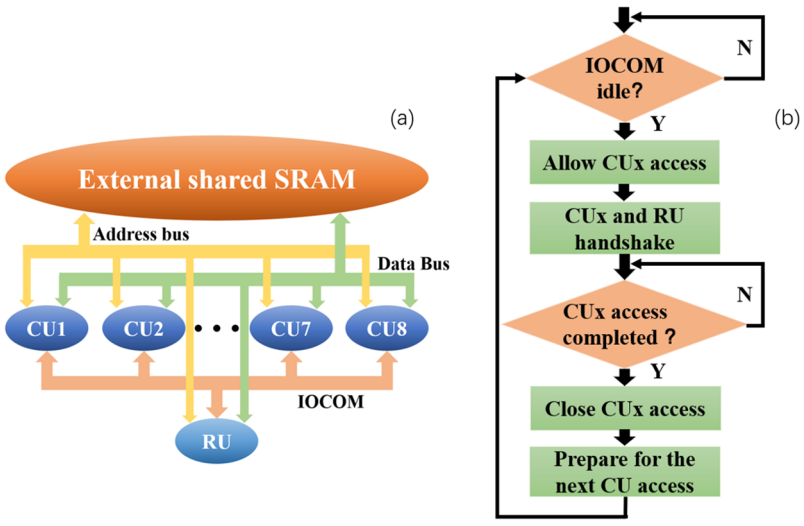

Figure 3. (**a**) Shared external resource access mechanism. (**b**) Mechanism to prevent access conflicts of shared external resources.

2.3. Inter-Chip and Board Communication Strategy

A single computing board cannot guarantee the real-time performance of the network when the neural network is very complex and computationally expensive. We need to use multiple computing boards to realize the decentralization of neural network tasks. After the task of a single neural network computing board is completed, it can communicate with the next computing board through USART and GPIO to complete the next distributed calculation. The next layer of the computing board sends a request to the current layer of the computing board. After the computing board of the current layer completes the

calculation task, it sends a signal to the computing board of the next layer. In the current layer, the RU accesses the data in the external RAM, writes the data packets that need to be distributed, and transmits the data to the RU in the next layer through the USART.

3. CNN Dimensionality Reduction Initialization and Distribution Mapping Mechanism

Convolutional neural network is a special artificial neural network, which has significant advantages in image processing and feature extraction. A typical CNN includes convolutional layers, pooling layers, and fully connected layers. The convolution process can extract the features of the image. The pooling layer is followed by the convolution layer. The amount of calculation is reduced by reducing the image size. Finally, the classification result is output through the fully connected layer. CNN needs to use backpropagation algorithms to train model parameters [23]. We mainly focus on the realization of forward calculation [24] of the CNN model completed by offline training. Under the hardware architecture of the many-core embedded parallel computing platform MEPP, CNN needs to perform split calculations and integrate the calculation results. There are two main factors considered in the model splitting process. One is to ensure computational efficiency. The forward calculation process of CNN requires many multiplication and addition calculations. The calculation amount of CNN tends to increase exponentially when the input image pixels are high. It takes a lot of time to choose serial calculation in a single chip. The second is limited resource storage space. Each layer of CNN needs to consume a lot of storage resources to save the weight parameters. Aiming at these two problems, the dimensionality reduction initialization and CNN distribution mapping mechanism are designed to solve the two problems of storage and calculation that limit the application of the model.

3.1. Overview of CNN Neural Networks

A convolution neural network is a kind of feedforward neural network including convolution operation, which is one of the most representative neural networks for deep learning. Convolutional operations have gained widespread attention in academia and industry due to their powerful ability to extract image features. Convolutional neural networks have made remarkable achievements in fields such as image classification, face recognition, object detection, and object tracking. In 1998, LeCun et al. proposed the classic LeNet-5 network, which has achieved success in handwritten digit recognition. Afterwards, convolutional neural networks received widespread attention in the field of computer vision. Structurally, the CNN mainly includes an input layer, hidden layer, and output layer. The hidden layer includes the convolution layer, pooling layer, full connection layer and activation function. In some complex CNNs, residual modules composed of the above hidden layers [23] and inception modules are also included. The convolutional layer is the core network layer of the entire CNN, used to extract the features of input image data. Each convolution core is connected to the region called the local receptive field in the upper layer, so as to learn the characteristics of this region. With the sliding window operation, the convolution kernel can learn the features of all receptive fields. In this process, the size, step size and filling of the convolution kernel can be manually set to obtain target feature maps of different sizes.

The pooling layer is another important component in CNN, usually following the convolutional layer to implement downsampling on the output feature map. The pooling layer can reduce model computation and memory usage while ensuring feature invariance. Common pooling operations include maximum pooling and average pooling. The fully connected layer usually appears at the end of the entire CNN. The feature map obtained after convolution is tiled and connected to each neuron in the fully connected layer. The calculation process of the fully connected layer is shown in the formulas below:

$$y = f(\sum_{i=1}^{N} w_i x_i + b) \quad (1)$$

where x_i is the i-th input neuron, w_i is the corresponding weight, b is the offset, and $f(x)$ is the activation function.

The calculation process of neural network is usually linear, and the activation function can introduce nonlinear characteristics into CNN, thus strengthening the learning ability of the network. Common nonlinear activation functions include Sigmoid, ReLU, hyperbolic tangent function (tanh function) and Softmax. Among these, the Softmax activation function is usually used on the network output layer to normalize the output values to obtain the probability of each output value. The formulas of these activation function are as follows:

$$Sigmoid(x) = \frac{1}{1+e^x} \tag{2}$$

$$\mathrm{Re}LU(x) = \begin{cases} x(x>0) \\ 0(x \le 0) \end{cases} \tag{3}$$

$$\tanh(x) = \frac{e^x + e^{-x}}{e^x - e^{-x}} \tag{4}$$

$$Softmax(x) = \frac{x}{\sum_1^j e^x} \tag{5}$$

3.2. Dimensionality Reduction Initialization

There are many problems in implementing distributed parallel computing of the CNN model on a multi-core embedded hardware platform. To implement the CNN model through a hardware platform, the first factor that needs to be considered is that of hardware storage resources. The memory size of STM32F4 series chips is 192 KB, including 128 KB RAM and 64 KB CCM RAM. In the calculation of each network layer, memory must be allocated for the input and output data of the layer, i.e., the network layer is initialized.

To address the challenge of limited memory resources, we propose a method that combines dynamic memory allocation and dimensionality reduction initialization. Dynamic memory allocation is a technique used to allocate and manage computer memory resources. When a memory application is executed, the required memory is allocated from the memory pool. After the data is processed, the memory resources are released and reclaimed. However, since the addresses of the two random-access memory (RAM) modules are not continuous, the actual RAM size that can support dynamic memory allocation is limited to 128 KB. Therefore, the amount of data declared through dynamic memory allocation at any given time must not exceed this limit.

As a single network layer typically involves a large amount of data, dynamic memory allocation alone is insufficient for initialization. To address this, we propose a method of dimensionality reduction and initialization. Typically, the feature map in a convolutional neural network (CNN) is represented as a three-dimensional matrix. Through dimensionality reduction initialization, we allocate memory only to the two-dimensional matrix when initializing a single network layer, i.e., a single feature map. Once the calculation of a single feature map is complete, we erase the data of the current feature map and write the data of the next feature map. In cases where the data volume of a single feature map is still too large, we can further reduce the dimensionality of the matrix allocation memory to allocate memory for a one-dimensional array. This approach allows us to optimize memory utilization while minimizing the impact on the performance of the CNN.

3.3. CNN Distribution Mapping Mechanism

Another key factor in implementing the CNN model on a multi-core embedded hardware platform is how to effectively map the CNN to multiple computing units. We propose a CNN distribution mapping mechanism based on the distributed characteristics of the hardware platform MEPP. It includes longitudinal layering, single-layer convolutional distribution mapping, single-channel image or feature map distribution after a single

convolution, and fully-connected layer distribution mapping. In this way, the entire network can be gradually dispersed.

As a serial network model, CNN itself has a hierarchical structure. According to the hierarchical structure of the CNN model, multiple inter-layer pipeline mapping methods can be used. The network can be split layer by layer, allowing each BEM to perform operations on a single network layer, or it can be mapped based solely on convolution, pooling, or full connectivity, allowing each BEM to perform forward inference on multiple network layers. Due to the fixed computing tasks of each CU, it is necessary to consider issues such as inter module data dependency and load balancing when performing CNN hierarchical mapping. FAM and BEM1 complete the first frame calculation to generate the classification convolution kernel and regression convolution kernel required for the cross correlation process, and transmit them to BEM2. At the same time, the location, length, width, and other information of the target in the first frame are stored in BEM3, and they need to be updated during the subsequent tracking process. FAM collects front-end images and performs corresponding calculations, and then waits for permission from the lower layer to transmit the calculation results. The cycle through this process starts from the collection of images. BEM1-3 also completes the corresponding loop calculation according to the requirements in the flowchart and transmits the calculation results to the lower layer. BEM3 makes the final judgment based on the calculation results.

The convolution process occupies more than 90% of the operations in CNN [8,25]. As the network layer with the largest amount of calculation, the convolutional layers need to be distributed to multiple computing units to ensure real-time requirement. There are two ways to map the convolutional layer. The first is to perform distribution mapping according to the number of input feature map channels, and the second is to perform distribution mapping according to the number of convolution kernels, i.e., the number of output feature map channels. Only by understanding how the feature map is convolved can the difference between the two distribution mapping methods be known. Take Figure 4 as an example. The input is a three-channel feature map. After convolution, a six-channel feature map is obtained. Therefore, it has to go through the convolution of six convolution kernels, and each convolution kernel is three-dimensional, corresponding to the input feature map of three channels. The three-channel feature map is convolved by a three-dimensional convolution kernel to obtain the output feature map of a single channel. After convolution of six three-dimensional convolution kernels, six-channel output feature maps are obtained. From this perspective, it can be said that each channel of the input feature map undergoes the convolution process of six convolution kernels, and it can also be said that each feature map of the output channel is convolved by three convolution kernels. In this way, the entire convolution process can be distributed to three calculation units and six calculation units. If the number of input feature map channels is the same as the number of convolution kernels, the time complexity of the distribution mapping method according to the number of input feature maps is smaller. In the process of network convolution layer by layer, in order to obtain more deep-level features, the number of feature map channels will gradually increase. Therefore, in this case, the distribution mapping according to the number of convolution kernels can make the network more decentralized.

The convolution of the single-channel feature map through a single convolution kernel is the usual two-dimensional convolution process. In the convolution process, a sliding window operation is required, as can be seen from the example in Figure 5. In Figure 5, different colored boxes divide the feature map into four equal parts and are assigned to four computational kernels. The dashed box represents the calculation tasks assigned to CU3, and displays the calculation process of CU3 using dashed lines.

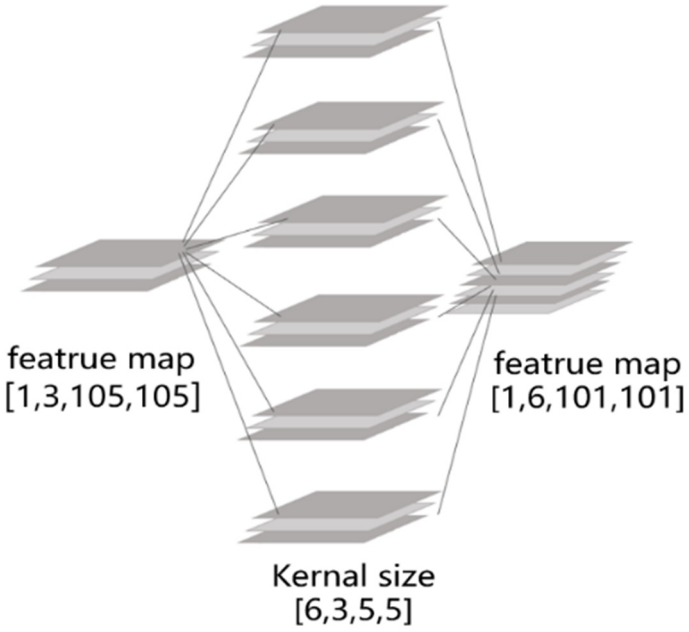

Figure 4. Feature map convolution process.

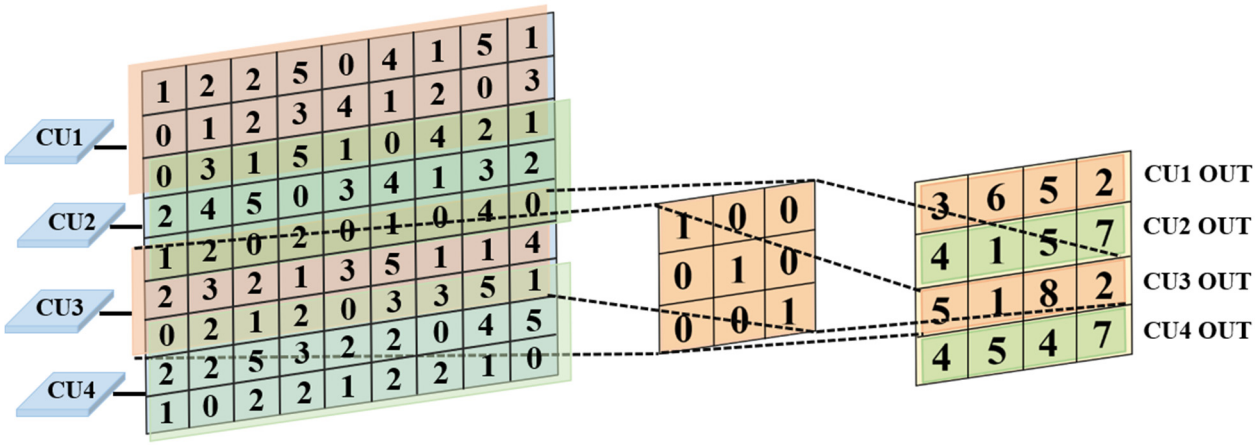

Figure 5. The distribution mapping process of a single-channel image or feature map after a single convolution.

The dashed lines indicate the start and the end of the calculation for CU3. In order to obtain a 4 × 4 feature map, four horizontal sliding windows are performed in the case of a step size of two. Under the premise of ensuring real-time performance, the single-channel feature map is split and mapped to multiple CUs. A single feature map can be divided into halves and quarters according to the number of horizontal sliding windows, i.e., the number of output channels. In this way, the distribution mapping of a single feature map for a single convolution is completed effectively.

The fully connected layer is divided into smaller neuron groups according to the number of neurons. Under the premise that each computing unit performs the same number of tasks, the fully connected layer is divided into equal parts according to the number of neurons. As shown in Figure 6, this is a fully connected layer with six neurons. The distribution can be mapped to two, three, and six computing units.

The CNN mapping mechanism can decentralize the CNN network effectively. After decentralization, the integration and horizontal collaboration of the computing output are realized through the shared resource access and storage mechanism in the layer. The

vertical cascade is completed between the layers through external communication between the groups.

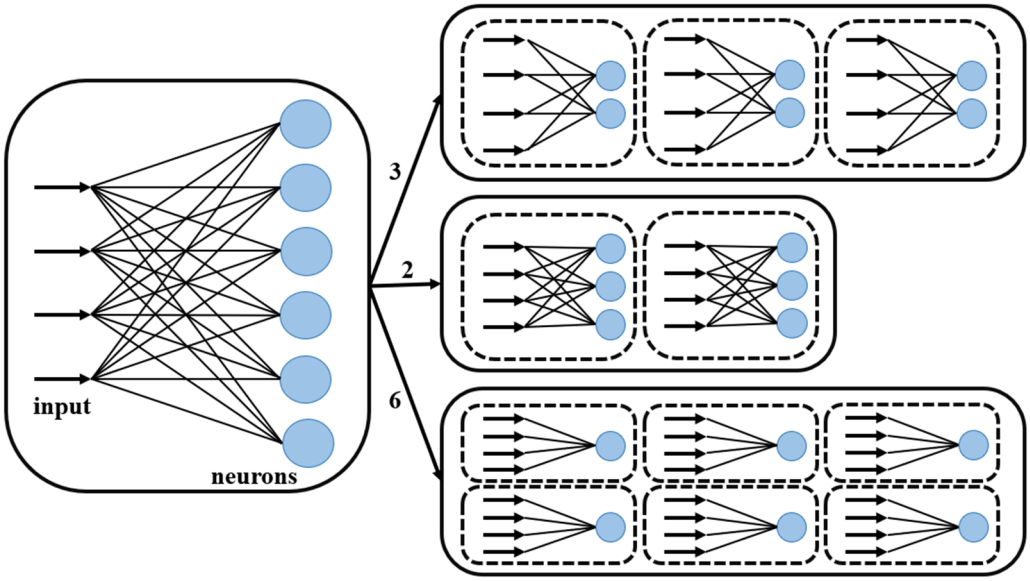

Figure 6. Fully connected layer distribution mapping process.

3.4. Calculation Optimization

In the process of neural network calculation, exponential functions may be involved in the nonlinear calculation process, such as sigmoid function and softmax function. The index calculation process consumes a lot of time. The chip we chose has a floating-point unit (FPU). It has excellent floating-point multiplication capability. Therefore, we replace the exponential operation in the standard library function with repeated multiplication according to the following formula.

$$e^x = \lim_{n\to\infty}(1 + x/n)^n. \tag{6}$$

The accuracy of this approximation method is gradually improved with the increase of n in the formula, and the calculation efficiency will decrease at the same time. In order to reduce the number of calculations as much as possible, we choose n to be converted to an exponential multiple of two. The reason is that, after such processing, when the model is converted into a difference equation to solve, the specific calculation steps are as follows:

$$\left.\begin{array}{l} x := 1 + x/2^k \\ x := x * x \\ \quad\vdots \\ x := x * x \end{array}\right\}(k\ times), \tag{7}$$

This calculation method can obtain a larger value of n through a small number of times of accumulation, so as to achieve both calculation efficiency and accuracy. This optimization mechanism improves the exponential calculation process. It greatly reduces the time overhead in the model calculation process.

In order to analyze the lightweight degree of deep separable convolution, the computational and parameter quantities of conventional convolution and deep separable convolution are calculated separately. The computational and parameter quantities of both are as follows:

$$FLOPs_{ratio} = \frac{C_{in} \times K \times K \times N \times N + C_{in} \times C_{out} \times K \times N \times N}{C_{in} \times K \times K \times C_{out} \times N \times N} = \frac{1}{C_{out}} + \frac{1}{K} \approx \frac{1}{K} \tag{8}$$

$$Parameters_{ratio} = \frac{C_{in} \times K \times K + C_{in} \times C_{out}}{C_{in} \times C_{out} \times K \times K} = \frac{1}{C_{out}} + \frac{1}{K^2} \approx \frac{1}{K^2} \quad (9)$$

From the above two equations, it can be seen that deep separable convolution significantly reduces the computational and parameter complexity of the model.

4. Experimental Verification and Result Analysis

We implement the LeNet-5 [26] model, which has different input sizes on this hardware platform. The LeNet-5 model of different input sizes is shown in Figure 7. In the implementation process, we design different experiments to verify the effectiveness of the dimensionality reduction initialization and CNN distribution mapping mechanism. Finally, we analyze the running time, resource consumption and power consumption.

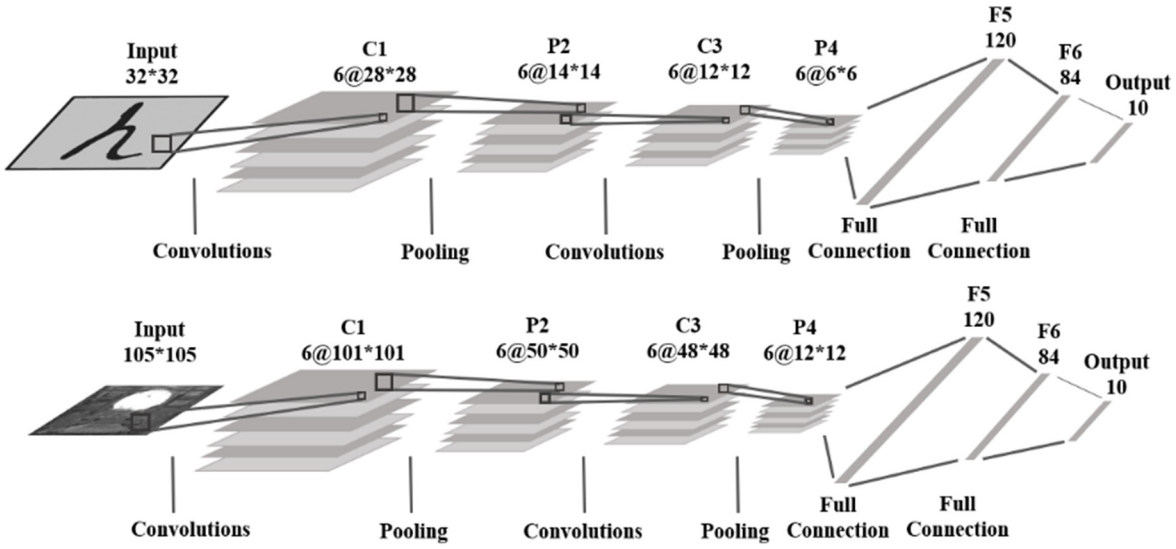

Figure 7. LeNet-5 model under different input sizes.

We verify the effectiveness of this method by comparing the memory occupied by the model before and after the dimensionality reduction initialization. Figure 7 mainly describes the processing method of LeNet-5 for feature maps. The input feature map is convolutionally processed through convolution layer C1, pooling layer P1, convolution layer C2, and pooling layer P2, and then linearized through two fully connected layers to obtain the output result. Figure 8 points out that, when the input image size is 32×32 or 105×105, each network layer needs to occupy memory during the calculation process. When the input image size is 32×32, a single chip can only provide memory for the data of a single network layer. When the input image size is 105×105, the amount of memory required for a single network layer data exceeds the maximum memory amount of the chip. Because of this, our experiment is mainly based on the image of 105×105.

When the input image size is 32×32, the chip memory can meet the requirements. We initialize the LeNet-5 network with an input image size of 105×105 for dimensionality reduction. This is mainly to initialize the dimensionality reduction of the convolutional layer and the pooling layer in the network. It can be seen from Figure 9 that in the 105×105 LeNet-5 model after dimensionality reduction, the memory occupied by the initialization of a single network layer is effectively limited to the chip memory size. The analysis results prove the effectiveness of the dynamic memory allocation and dimensionality reduction initialization method. This method can completely solve the problem of insufficient memory.

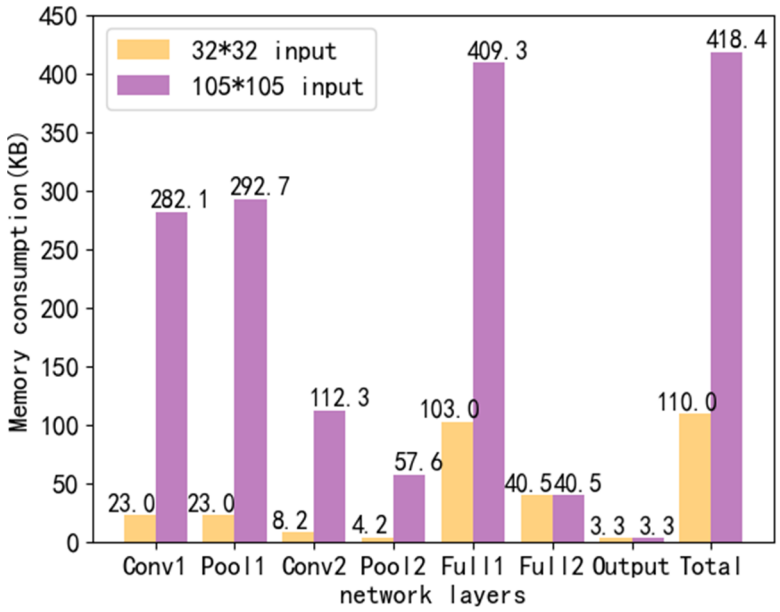

Figure 8. The memory usage of each network layer of the LeNet-5 model under different input sizes.

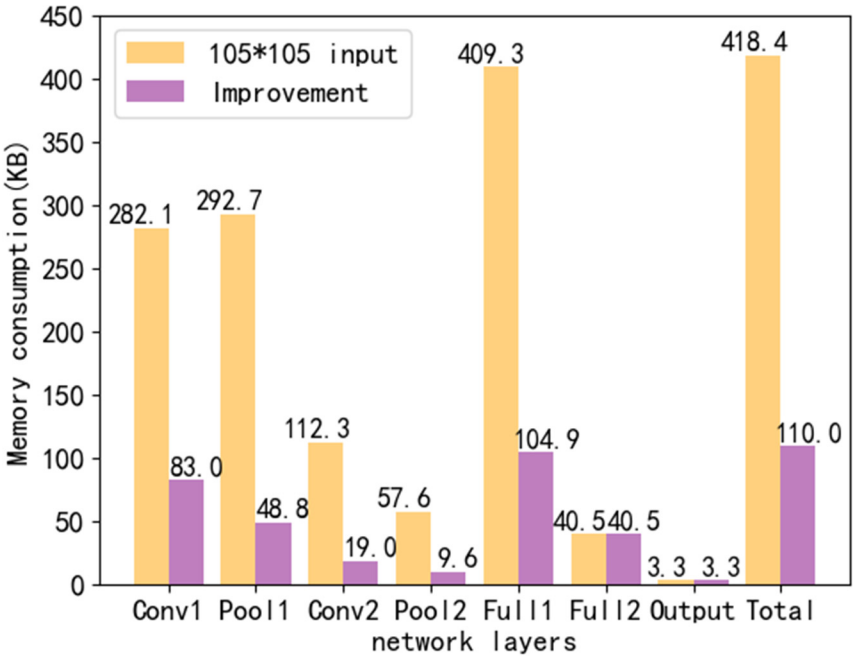

Figure 9. The comparison of the memory usage of each network layer before and after the dynamic allocation of memory and dimensionality reduction initialization of the LeNet-5 model with an input image size of 105 × 105.

Without considering real-time performance, we use a single core to implement LeNet-5 models of different sizes. Figure 10 indicates the time consumption. When the image input size is 105 × 105, the model running time exceeds 1s, which can no longer fulfil the real-time requirement. Among them, the convolutional layer occupies most of the calculation of the entire network.

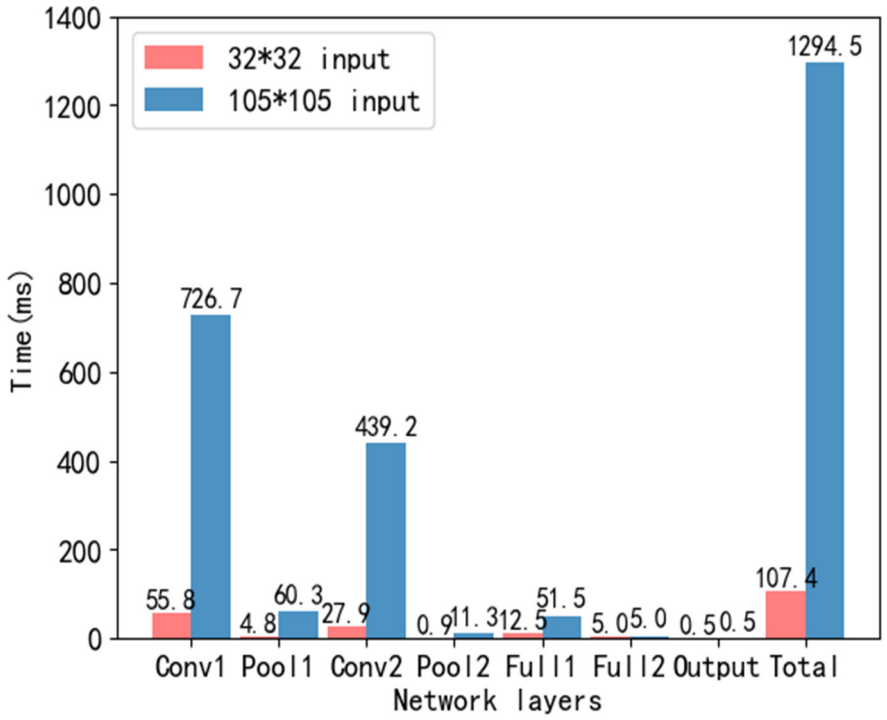

Figure 10. The network layers and total time consumption of the LeNet-5 model under different input sizes.

Next, we verify the effectiveness of the CNN distribution mapping mechanism. Respectively, the convolutional layer, pooling layer or fully connected layer are used as the scattered points for mapping. In addition, every layer of the entire network is decentralized. Compare the effects of various mapping methods. Figure 11 points out the time consumption of different distribution mapping methods. The method of board data transmission uses SPI (Serial Peripheral interface, serial peripheral interface). It can be seen that the running time of the model is significantly shortened after the distributed mapping of the network. The distributed mapping method that takes the pooling layer as the dispersion point consumes the shortest time. The reason is that the amount of calculation in the pooling layer is not large. Secondly, the pooling layer reduces the resolution of the feature map to obtain features that are not spatially deformed. The amount of data transmitted during board communication is greatly reduced, so the board communication time is shorter. Therefore, when the convolutional layer is followed by the pooling layer, taking pooling as the dispersion point, each unit completes a layer of convolution and pooling. This can achieve the optimal mapping.

The effect of distribution mapping is analyzed according to the number of convolution kernels and distribution mapping according to the number of input feature maps. The $6 \times 50 \times 50$ feature map convolution is implemented in two ways to obtain a $6 \times 48 \times 48$ feature map. Finally, it takes 3.5 ms longer for the distribution mapping according to the number of convolution kernels than the distribution mapping according to the number of input feature map channels. When the number of input feature map channels and the number of convolution kernels are the same, the mapping is performed in two ways, and the calculation amount in a single calculation board is the same. The calculation time should be equal. The reason for the time-consuming difference between the two mapping methods is that the data needs to be read from the external RAM when the feature map is initialized. However, the distribution mapping according to the number of convolution kernels requires multiple initializations, i.e., multiple accesses to the external RAM. Therefore, when the number of input feature map channels is equal to the number of

convolution kernels, the time complexity of distribution mapping according to the number of input feature maps is smaller.

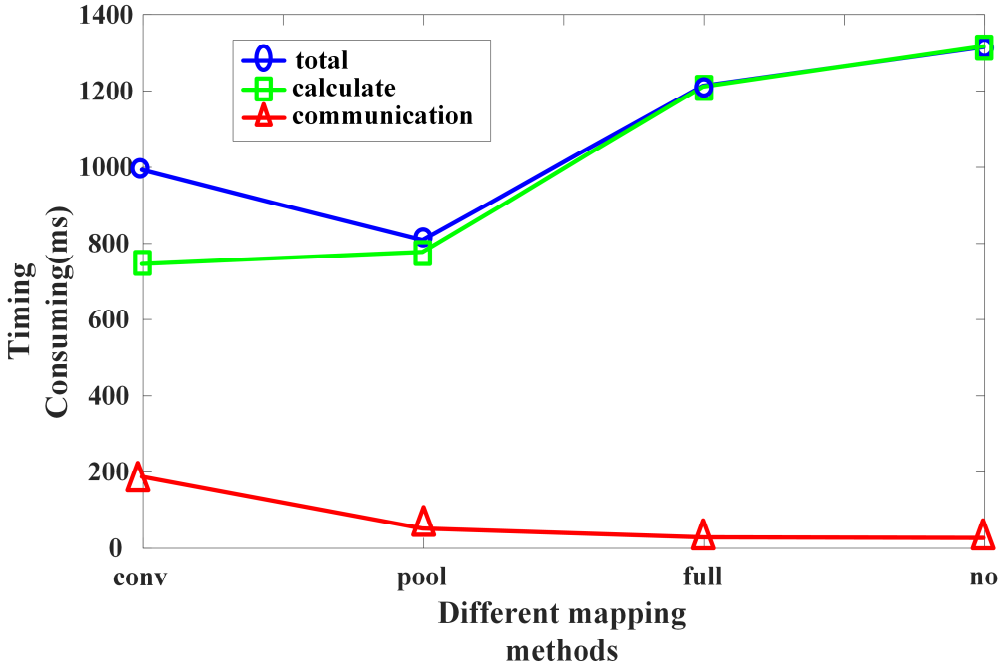

Figure 11. Time consumption of the network layer in different mapping methods.

We have tested the recognition accuracy of multi-core LeNet-5 through a large number of experiments. Under the normal use of the sigmoid function, the recognition accuracy of LeNet-5 is 88.2%. After simplifying the sigmoid function calculation, the recognition accuracy of LeNet-5 for the same conditions and samples decreases to 83.3%. The accuracy has decreased by 4.9%, but according to the above equation the calculation amount has been reduced to $\frac{1}{4}$ of the pre simplification level. This 4.9% decrease in accuracy results in a savings of around 25% in computational complexity. This is very meaningful for STM32, which has far less computing power and memory than GPUs. Our embedded platform expects to achieve lightweight and portable CNN operations, while ensuring an accuracy rate of 80%, and striving for lightweight and efficient models and calculations as much as possible. Balancing the two, as the recognition accuracy is still above 80%, we believe that the simplified sigmoid function is more suitable for our platform.

We complete the entire improved LeNet-5 model mapping process through layer-by-layer mapping, convolution kernel mapping, single convolution or fully connected distributed mapping. The specific distribution details are shown in Table 1. The task used 26 chips, 22 of which are used for the CNN network part. The C1 and P1 layers are scattered according to the size of the convolution kernel, and then a single feature map is scattered into two parts and mapped into 12 computing units. We adopted an equal division method during the experimental process. After the grouping calculation is completed, merge again to obtain the convolution calculation results. The C2 and P2 layers only need to be mapped to six computing units according to the size distribution of the convolution kernel to meet the requirements. The calculation amount of the fully connected part is relatively small, and initialization through dimensionality reduction can be completed by a single calculation unit.

Table 1. CNN distribution implementation details.

Layer	CU	BU	Input	Output
C1 P1	12	2	105 × 105 × 1 100 × 100 × 6	101 × 101 × 6 50 × 50 × 6
C2 P2	6	1	50 × 50 × 1 48 × 48 × 6	48 × 48 × 6 12 × 12 × 6
F1 F2 Output	1	0	864 × 1 120 × 1 84 × 1	120 × 1 84 × 1 10 × 1

The CU calculates the time loss of a synchronization operation and inter core communication in sequence. Based on the average polling time of 100,000 word iterations, we measured the average polling time of the calculation module to be 488 µs. The proportion of polling time to the total calculation time cycle is 39.8%. The experimental results indicate that a large number of neurons can further reduce the proportion of polling time and improve the computational efficiency of the platform.

Figure 12 is a comparison of the time consumption of single-core and multi-core implementation of improved LeNet-5. The model is distributed to the neural network parallel computing hardware platform MEPP through the dimensionality reduction initialization and the CNN distribution mapping mechanism, which effectively shortens the model running time.

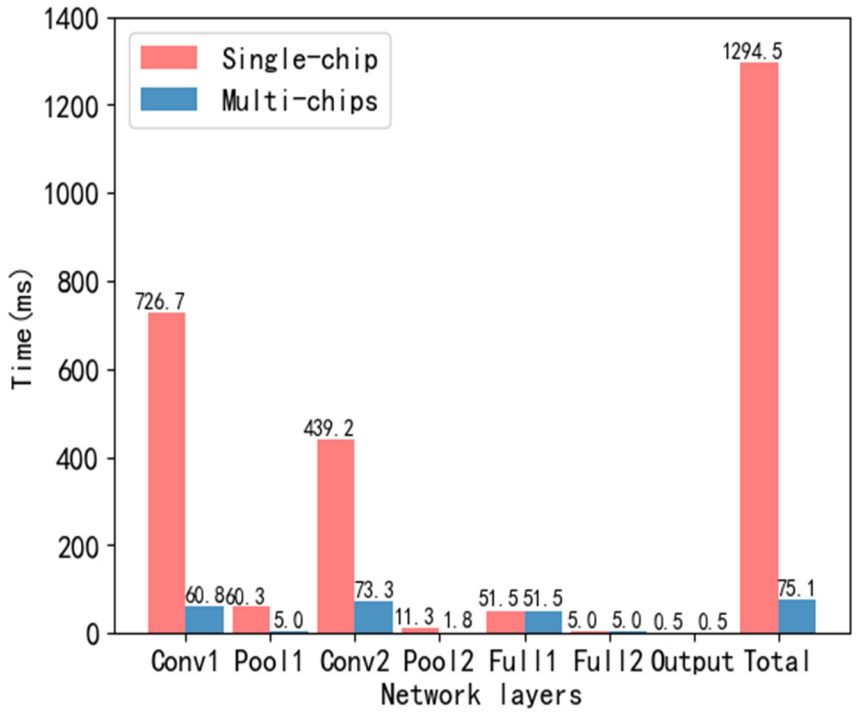

Figure 12. Comparison of single-core and multi-core implementation time consumption.

Table 2 lists the power consumption of the CU responsible for different functions in the platform MEPP. The power consumption of the system is at a relatively low level. This is one of the important advantages of the platform in specific application scenarios.

Table 2. Power consumption of the embedded platform MEPP.

CU Function	Working Current	Single Chip Power Consumption
C1/P1	69.8 mA	0.230 W
C2/P2	67.0 mA	0.206 W
F1/F2/Out	62.5 mA	0.221 W
RU	46.4 mA	0.153 W

Table 3 shows the resource consumption of a single chip. The fully connected layer is still the network layer that consumes the most memory. These analysis results characterize the practical capabilities of this system in CNN applications. The good scalability brought by its modular organizational structure of MEPP also lays the foundation for the realization of more complex algorithms and larger-scale tasks.

Table 3. Storage resources overhead of the embedded platform.

CU Function	Single Chip RAM Overhead	Single Chip FLASH Overhead
C1/P1	62.94 KB (49.17%)	37.44 KB (3.66%)
C2/P2	19.24 KB (15.03%)	37.65 KB (3.67%)
F1/F2/Out	104.93 KB (81.99%)	37.61 KB (3.67%)
RU	58.88 KB (46.00%)	10.92 KB (1.07%)

We analyze the time complexity of convolution operations in a lightweight CNN using techniques such as the sliding window method to count the number of operations required for each convolutional layer. We analyze the memory requirements of lightweight CNNs and how they scale with the number of parameters and layers in the network.

5. Conclusions

Edge computing can decompose large-scale neural networks into multiple real-time computing tasks, greatly saving the energy and time consumption of computing, but the implementation of edge computing in traditional embedded single core microcontroller is not easy. We envisage improving the CNN algorithm to decompose the computational tasks of CNN, realizing part of the tasks of edge computing in a single core microprocessor, and realizing lightweight edge computing with less time cost and resource cost through parallel cooperation among multiple single core microcontrollers. In this paper, in response to the requirements of edge computing for power consumption and real-time performance, we build an ARM-based embedded parallel computing hardware platform MEPP. In order to complete the front-end image acquisition tasks such as image classification and target tracking, a front-end acquisition and computing board hardware architecture is designed. Drawing on the idea of time division multiplexing, we propose a multi-chip shared external static random access memory (SRAM) access mechanism. This method solves the communication problem between chips. Then, we propose a method of dimensionality reduction initialization to solve the problem of on-chip resources. According to the hierarchical structure of the convolutional neural network and the characteristics of the sliding window operation in the convolution process, a multi-level CNN distribution mapping mechanism is designed. The most time-consuming exponential function calculation in the nonlinear calculation process is optimized, and the exponential function calculation speed is greatly improved at the cost of reducing a small part of the calculation accuracy.

The experiments verify that the neural network parallel computing hardware platform can implement the CNN model. It has the advantages of low power consumption, scalability, and low cost. At the same time, the effectiveness of dimensionality reduction initialization and the CNN distribution mapping mechanism in real-time realization of convolutional neural network is verified. In the follow-up work, we will consider changing

the communication method and control chip of the board. Faster data transmission communication methods and higher-end chips can better ensure real-time performance. On the basis of this architecture, we can replace the STMF407 series, which is currently used on the platform and has relatively low main frequency and computing power, with more powerful chips to achieve more efficient computing functions. Based on these experiments, we will iteratively upgrade STM32 in the embedded platform architecture, replacing STM32 with more advanced embedded chips such as Raspberry Pie. In future experiments, we will have the conditions to conduct experiments using these modern CNN networks.

The real-time of model computation is one of the core issues of convolutional neural network hardware acceleration. Real time performance is mainly constrained by computational power and inter core communication latency. In this experiment, we have demonstrated the effectiveness of achieving high-speed communication through shared RAM. However, the performance of hardware platforms has not been fully utilized, and it is still necessary to discuss methods to achieve higher computational power in future work, introducing a more powerful kernel and larger off-chip storage space under the current architecture, and expanding more diverse means of off chip communication. In future work, we will further expand the platform's application scenarios by adding hardware resources, such as external cameras and infrared cameras, to achieve target recognition and tracking functions. At the same time, the platform can only achieve large-scale CNN operations and has a narrow application range. However, other physiological neural networks can also perform distributed computing based on the structure of our platform. In the future, we will investigate the performance of other physiological neural networks on the platform. On the basis of CNN experiments, we also conducted experiments on pulse neural network SNN on the platform. In addition to designing a multi-core system, we also designed a lightweight pulse neural network for the multi core system to optimize the model deployment process, making the network design more hardware friendly. In future research, we will continuously optimize our platform and the networks it adapts to from two aspects: reduce network size and compression weight parameters, in order to achieve richer parallel computing functions.

Author Contributions: Conceptualization, X.W. and G.Y.; Methodology, writing—original draft preparation, writing—review and editing, L.J., G.L. and M.L.; validation, investigation, X.W. and G.Y.; supervision, project administration, funding acquisition, X.W. and M.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the National Natural Science Foundation of China (grant numbers 62171312) and by the Tianjin Municipal Education Commission scientific research project (grant number 2020KJ114).

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I.; et al. A View of Cloud Computing. *Commun. ACM* **2010**, *53*, 50–58. [CrossRef]
2. Shi, W.; Sun, H.; Cao, J.; Zhang, Q.; Liu, W. Edge Computing-An Emerging Computing Model for the Internet of Everything Era. *J. Comput. Res. Dev.* **2017**, *54*, 907–924. [CrossRef]
3. Satyanarayanan, M. The Emergence of Edge Computing. *Computer* **2017**, *50*, 30–39. [CrossRef]
4. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge Computing: Vision and Challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
5. Dayhoff, J.E. Neural Network Architectures: An Introduction. *Choice Rev. Online* **1991**. [CrossRef]
6. LeCun, Y.; Bengio, Y. Convolutional Networks for Images, Speech, and Time Series. *Handb. Brain Theory Neural Netw.* **1995**, *3361*, 1995.
7. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]

8. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015-Conference Track Proceedings, San Diego, CA, USA, 7–9 May 2015.
9. Long, J.; Shelhamer, E.; Darrell, T. Fully Convolutional Networks for Semantic Segmentation. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; Volume 7.
10. Farabet, C.; Couprie, C.; Najman, L.; Lecun, Y. Learning Hierarchical Features for Scene Labeling. *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *35*, 1915–1929. [CrossRef] [PubMed]
11. Zhang, Y.; Chan, W.; Jaitly, N. Very Deep Convolutional Networks for End-to-End Speech Recognition. In Proceedings of the ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing-Proceedings, New Orleans, LA, USA, 5–9 March 2017.
12. Gidaris, S.; Komodakis, N. Object Detection via a Multi-region and Semantic Segmentation-Aware CNN Model. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1134–1142. [CrossRef]
13. Redmon, J.; Farhadi, A. YOLO9000: Better, Faster, Stronger. In Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, 21–26 July 2017; Volume 2017-January.
14. Bertinetto, L.; Valmadre, J.; Henriques, J.F.; Vedaldi, A.; Torr, P.H.S. Fully-Convolutional Siamese Networks for Object Tracking. In Proceedings of the Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Melbourne, VIC, Australia, 1–6 December 2016; Volume 5866 LNAI.
15. Bao, L.; Wu, B.; Liu, W. CNN in MRF: Video Object Segmentation via Inference in a CNN-Based Higher-Order Spatio-Temporal MRF. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018.
16. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the Inception Architecture for Computer Vision. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; Volume 2016-December.
17. Fukagai, T.; Maeda, K.; Tanabe, S.; Shirahata, K.; Tomita, Y.; Ike, A.; Nakagawa, A. Speed-up of Object Detection Neural Network with GPU. In Proceedings of the International Conference on Image Processing, ICIP, Athens, Greece, 7–10 October 2018.
18. Huang, J.; Wang, T.; Zhu, X.; Wei, M.; Wu, T.; Wu, X.; Huang, M. A Parallel Optimization of the Fast Algorithm of Convolution Neural Network on CPU. In Proceedings of the 10th International Conference on Measuring Technology and Mechatronics Automation, ICMTMA 2018, Changsha, China, 10–11 February 2018; Volume 2018-January.
19. Saha, S.S.; Sandha, S.S.; Srivastava, M. Machine Learning for Microcontroller-Class Hardware: A Review. *IEEE Sens. J.* **2022**, *22*, 21362–21390. [CrossRef] [PubMed]
20. Deng, L.; Li, G.; Han, S.; Shi, L.; Xie, Y. Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey. *Proc. IEEE* **2020**, *108*, 485–532. [CrossRef]
21. Ian, G.; Yoshua, B.; Aaron, C. *Deep Learning, Adaptive Computation and Machine Learning Series*; MIT Press: Cambridge, MA, USA, 2016; ISBN 9780262035613.
22. Messaoud, S.; Bouaafia, S.; Maraoui, A.; Ammari, A.C.; Khriji, L.; Machhout, M. Deep convolutional neural networks-based hardware-software on-chip system for computer vision application. *Comput. Electr. Eng.* **2022**, *98*, 107671. [CrossRef]
23. Jia, Y.Q.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; Darrell, T. Cafe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia (MM'14). Association for Computing Machinery, New York, NY, USA, 3–7 November 2014; pp. 675–678. [CrossRef]
24. Zhang, C.; Li, P.; Sun, G.; Guan, Y.; Xiao, B.; Cong, J. Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks. In Proceedings of the FPGA 2015-2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2015.
25. Chen, Y.H.; Emer, J.; Sze, V. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In Proceedings of the 2016 43rd International Symposium on Computer Architecture, ISCA 2016, Seoul, Republic of Korea, 18–22 June 2016; pp. 367–379.
26. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proc. IEEE* **1998**, *86*, 2278–2323. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.