engineering proceedings

MDPI

Proceeding Paper

Research on an Embedded Intelligent Computing Technology †

Pengtao Su * and Yihan Liu

AVIC Xi'an Aeronautics Computing Technique Research Institute, Xi'an 710065, China; 18710464537@163.com

* Correspondence: lyhsports@hotmail.com

† Presented at the 2nd International Conference on Green Aviation (ICGA 2024), Chengdu, China, 6–8 November 2024.

Abstract

In this paper, we study the realization in the embedded platform of an Artificial Neural Network (ANN) to explore the feasibility of a lightweight neural network. The purpose is to provide theoretical support for the intelligent information processing system based on a neural network under the airborne embedded environment, and then to support the intelligent demand of the airborne system in the future complex battlefield. By analyzing the application prospect of ANN in the aviation field, this paper will design an embedded hardware platform for UAV autonomous intelligence information processing tasks, and realize a convolution neural network (CNN) on it, using an image recognition task to train and test.

Keywords: intelligent computing; artificial neural networks; embedded system; FPGA

1. Background

With the continuous development of computer architecture and related hardware level, the computational power and the scale of data processing are constantly improving, thereby providing advanced means for human beings to solve problems with clear characteristics, strict descriptions with mathematical or logical language, and promoting the development of several disciplines. However, information processing problems in nature are not only structured problems, but also unstructured ones in many cases [1]. The latter are difficult to solve by direct computer programming. With society's increasing demand for intelligence, upgrading the level of cognition and decision making of traditional computers to meet the needs of generalized artificial intelligence (or strong AI) has become an urgent problem.

The future war will be multi-military integrated combat carried out by sea, land, air, satellite, radio and other multi-dimensional spaces; the battlefield environment is becoming increasingly complex, and the situation is changing rapidly. As a consequence, in the field of aviation, the needs of aircraft for intelligent airborne systems are also increasingly higher and higher. Despite the use of manned or unmanned aircraft, there is an urgent need for a highly intelligent information processing system to realize real-time analysis of a large amount of battlefield information; a highly intelligent control system is needed to realize the accurate operation of each piece of airborne observation equipment, weapon equipment, flight control equipment, etc. For the above demand for artificial intelligence in the aviation field, it is found that ANN-based intelligent computing technology shows good application prospects [2]. Combining it with traditional airborne computer systems to study airborne real-time intelligent decision-making (assistance) systems will help to solve a series of problems faced by manned/unmanned airplanes in the future. However, most of the existing ANNs run on platforms such as desktop servers, workstations, supercomputers

check for updates

Academic Editors: Weihong Zhang, Shichun Yang, Leiting Dong, Xungang Diao, Shijun Yin, Lixin Guan, Zuxi Xia and Xue Zhang

Published: 18 February 2025

Citation: Su, P.; Liu, Y. Research on an Embedded Intelligent Computing Technology. *Eng. Proc.* **2024**, *80*, 32. https://doi.org/10.3390/engproc2024080032

Copyright: © 2025 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/).

and computer clusters, which have huge volume and power consumption, and are far from the requirements of airborne-embedded environments. Therefore, this paper will study the implementation of ANNs on an embedded platform, explore the feasibility of a lightweight embedded neural network, in order to provide theoretical support for improving the intelligence level of airborne computer systems and truly realizing intelligent information processing in the future.

2. FPGA-Based Implementation of CNNs

2.1. Data Format and Accuracy

The selection of data format is one of the key issues to be considered when performing inter-platform algorithm migration. For neural networks, high-precision floating-point arithmetic implies fewer quantization errors, which can lead to more accurate computation results. But it requires more hardware resources and computing time, and the implementation of floating-point arithmetic on FPGAs is also a considerable challenge. Lower data precision will have the opposite effect. Reducing the number of data bits can effectively simplify the design of ANNs, but low precision will lead to an increase in the error, making the convergence of the neural network worse, and it may even appear to oscillate. According to the research results of related papers, when implementing ANNs on FPGAs, 16-bit fixed-point decimals are the minimum standard to meet the computing accuracy of neural networks. When people study the hardware implementation of neural networks, a 16-bit to 32-bit fixed-point decimal format is generally chosen [3].

Considering that, in the computing process of the implemented CNN in this paper, the data of the feature map is processed by the Sigmoid function, and the integer bit data are small, for the feature map data, the 16-bit fixed-point decimal format is used, which contains 1 sign bit, 5 integer bits and 10 decimal bits. Since the data accuracy of 10 decimal bits is only 0.001, this paper uses a PC to assist in the learning process, and directly applies the parameters of the completed network training process to the hardware implementation. In order to avoid the introduction of excessively large error, which could affect the recognition results, the weights of the convolution kernel data use the format of 24-bit fixed-point decimals, with 1 symbol, 5 integer, and 18 decimal places, which ensures that the convolution kernel is able to play the role of feature extraction and classification.

Therefore, during the experiment, the input data format of the convolutional computing unit is (1, 5, 10) signed fixed-point decimal, and the data format of the convolution kernel is (1, 5, 18) signed fixed-point decimal. Considering that the calculation process of fixed-point decimal multiplication is the same as that of integer multiplication, and only the shift of the decimal point needs to be noted, the signed multiplication operation is therefore directly realized using the DSP-based IP core. Then, the input of the multiplier is 16-bit and 24-bit, respectively, and the output is 40-bit, of which the lower 28 bits are decimal bits. Since the precision of 18 decimal bits is already sufficient, this paper directly discards the lower 10 bits of the output result when implementing the multiplier. The IP configuration information of the multiplier is shown in Figure 1.

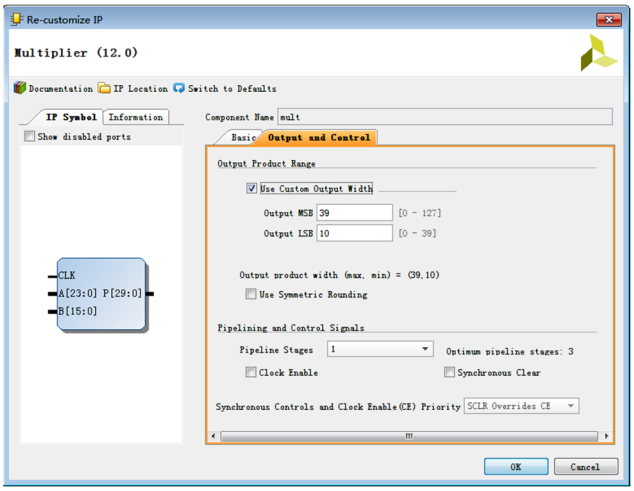


Figure 1. IP configuration of multiplier.

2.2. Implementation of Neuron Circuit

Neurons are the node units that constitute a neural network. When a neural network is implemented in the form of hardware, the first thing to consider is how to implement the neurons. From the introduction of the neuron model in Section 2.1, it is clear that the most basic neuron circuits can be constituted using multipliers, accumulators and activation function calculation units. In the computing process of a neural network, the multiplication operation occupies a great proportion, and many researchers take the optimization of multiplication operation as a breakthrough point to improve the computing speed of neural networks [4]. For example, by restricting all integer powers 2 of weights in the network, shifting is used instead of multiplication, or distributed algorithms are used to improve the utilization of hardware resources during multiplication operation. Thanks to the development of hardware technology, this paper uses the DSP hard core integrated inside the FPGA chip to carry out signed multiplication calculation, which already has a very good computational speed. Therefore, in hardware design and optimization, the multiplication calculation process is not studied, but through sacrificing the hardware area, a large number of multipliers and summators are used to improve the parallel computing ability of the circuit, and to enhance the computing efficiency of the neural network.

For CNNs, the data structure of each layer is no longer a one-dimensional vector, but a feature map of a number of dimensions, and simple sequential computing is inefficient.

Considering that each convolutional kernel has to perform convolutional computing with the whole input feature map, there exists a large amount of data reuse, and constructing a suitable pipeline structure can effectively reduce the number of reading and writing data and improve the computing speed [5].

A convolutional computing acceleration scheme, by constructing a 3×3 scale convolutional operation unit containing 9 multipliers and 8 summators, realized parallel implementation of the multiplication and addition process in the convolutional computing, so that the convolutional computing unit could reach a maximum of one data output per cycle, to greatly improve the computing efficiency.

In this paper, when the convolutional computing unit circuit is designed, the parallel computing features of FPGAs are utilized to fully unfold the convolutional computation. The size of the convolution kernel used in this paper is 5×5, so that 25 multipliers and 24 summators are used to realize the convolution calculating function. As shown in Figure 2, the weight information of the convolution kernel is pre-stored in BRAM and read into registers before the start of the calculation, thereby ensuring that the 25 multipliers can execute the calculation in parallel. The summators are distributed into a five-layer

tree structure, which generates a total of 5 clock cycles of delay. Five input data interfaces correspond to the five rows of data that are currently undergoing convolutional computing, and one data point is entered in each clock cycle. After completing one row of computing, the data will be shifted up one row as a whole by the controller, so when the convolutional computing unit first starts to work, it needs five clock cycles to fill the data; during the computing process, the filling of the new row of data will be synchronized with the exit of the previous data [6].

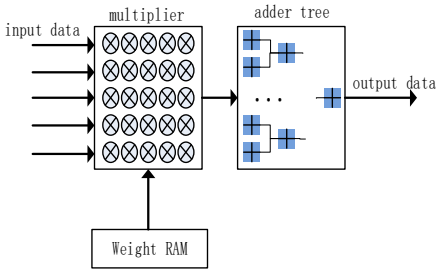


Figure 2. Schematic diagram of convolutional computing unit.

The simulation results of the convolutional computation unit are shown in Figure 3. In order to facilitate the presentation of the calculation results, here in this paper, all the input data are set to 1, and the weights of the convolution kernel are also 1. According to the fixed-point decimal format selected above, their hexadecimal representations are 0×0400 and 0×040000, respectively; the correct value of convolutional computing should be 25, use the data format of 1 bit sign bit, 10 integer bits, 18 decimal bits, and the hexadecimal representation is 0×00640000.

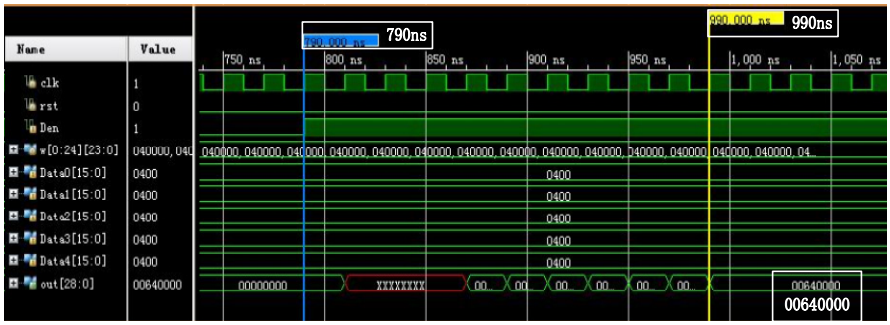


Figure 3. Simulation of convolutional computation unit.

The clock period used in the simulation is 20 ns. As can be seen from Figure 3, at 790 ns, the enable signal Den changes to 1, the convolutional computing unit starts to work; after 5 cycles of the data filling process and 5 cycles of delay in the addition tree, the first valid output is generated at 990 ns.

In CNN computing, after the convolutional computing unit completes one line of computing, the input data are shifted up by one line as a whole to start the next line of computing, Figure 4 shows this sequential upward shift of data. For example, the value of Data4 is 0×0150 at 2050 ns in the figure, and after 28 cycles (560 ns), the value of Data3 becomes 0×0150. With sufficient resources, a separate convolutional computing unit is configured for each convolutional kernel to ensure full utilization of the parallel computing characteristics of the network. Since different convolutional computations in the same layer are performed in parallel, when the output feature map is associated with multiple input feature maps, it is only necessary to add a number of summators after the convolutional computing unit, and the final output feature map is obtained after the activation function computing unit.

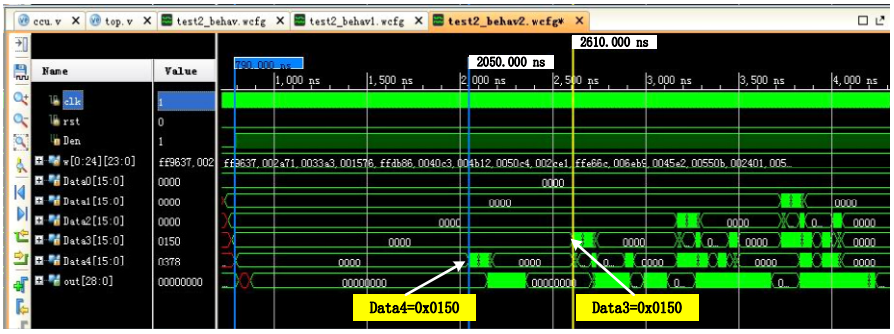


Figure 4. Convolutional layer computation simulation.

2.3. Selection and Realization of Activation Functions

The activation function is used to realize the nonlinear transformation from the input space to the output space in ANN, which reflects the information processing characteristics of the neural network, being an important part of the neural network model.

Hyperbolic tangent function $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and Sigmoid function $f(x) = \frac{1}{1+e^{-kx}}$ are common activation functions in ANNs [7]. But since FPGA circuits cannot directly compute the exponential function, this paper chooses to use an approximation scheme for the solution when performing the hardware implementation. Common approximation schemes are the look-up table (LUT) method, segmented linear approximation method, the CORDIC-based method and table-driven linear interpolation method. The LUT method pre-stores the partial results of the activation function in the lookup table, which theoretically can achieve high accuracy; the cost is a huge storage space, and the lookup speed is also a limitation. The segmented linear approximation method uses a linear function $y = ax + b$ to perform approximate computing for a part of the activation function, of which the computing efficiency is high, but usually the approximation is not high. The CORDIC-based algorithm is more accurate, but requires a complex arithmetic module, and the arithmetic time is longer. The table-driven linear interpolation method is a combination of the first two methods. The nonlinear function used as an activation function usually has salient changes only in the vicinity of the function center value; this part is stored in the lookup table to obtain a higher accuracy. The linear function is utilized for the part changing slowly outside the center of the region for approximation, in order to save storage space and obtain higher speed. But the implementation of this method is relatively more complex [8].

In this paper, the Sigmoid function is chosen as the CNN activation function, which is simpler than the hyperbolic tangent function, but still can achieve good nonlinear mapping effect; the approximation computing scheme chooses the segmented linear approximation method.

When designing the parameters of the segmented linear approximation method, this paper tries different segmented intervals, and plots the function curves through MATLAB to determine whether the fitting effect meets the computing needs. Through experiments, this paper finds that the Sigmoid function transforms more obviously between the independent variable intervals $[-6, 6]$. When the independent variable is smaller than -6, the function value tends to be close to 0; when the independent variable is larger than 6, the function value tends to be close to 1. Therefore, when carrying out the segmented linear approximation, this paper mainly focuses on the independent variable interval $[-6, 6]$ for the study. After comparing a variety of segmentation schemes, this paper finally chooses to divide the independent variable of the function into seven segments according to the nodes $[-3, -2, -1, 1, 2, 3]$. The function curve is shown in Figure 5. As can be seen from the figure, the scheme achieves very good approximation results, and the function curve of

the segmented linear approximation and the Sigmoid function curve are almost completely overlapping, and the error is only slightly larger near the segmented nodes −3 and 3.

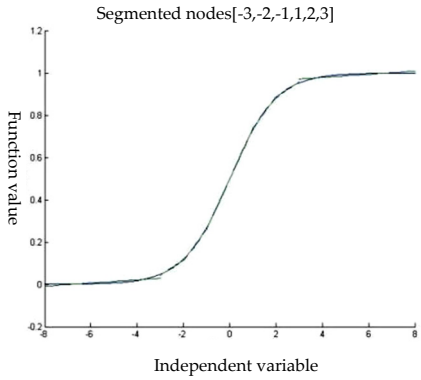


Figure 5. Function curve.

3. System Performance Testing and Analysis

3.1. Experimental Test Environment and Evaluation Methods

Based on the design of the embedded intelligent information processing hardware platform and related key technologies introduced in the previous section, this paper designs and produces an embedded intelligent information processing hardware platform to realize a CNN for handwritten digit recognition. In this section, the performance of the implemented CNN is tested and evaluated based on this hardware platform.

3.1.1. Hardware and Software Development and Test Environment

The software development tools used in this paper are MATLAB 2010, Visual Studio 2010, Xilinx SDK 2015.4; the hardware development tool is Vivado 2015.4; the PC model used for development and testing is ThinkCentre M8500t-N000 (Lenovo, Beijing, China). The main performance parameters are shown in Table 1. The processing unit is a dual-core 3.6 GHz I7 CPU.

Table 1. Performance parameters of PC.

Operating System	Windows 7 Ultimate
Processing unit	Intel®Core™ i7-4790 CPU @ 3.60 GHz 3.60 GHz
Memory	4.00GB
System type	64-bit OS
Power consumption	280 W

The embedded intelligent information processing hardware platform designed and produced in this paper takes two XC7Z045 chips as the core, and externally realizes a VGA input interface, DVI input/output interface, Gigabit Ethernet interface, RS232 interface and JTAG interface.

3.1.2. Evaluation Methodology

Given that this paper chooses the AP SoC as the core device of the hardware module, the chip integrates a dual-core ARM processing unit and can directly run basic C programs. In this paper, the research first implements a C language-based CNN, and uses the program to complete the training and testing process of the neural network on a PC, evaluates the recognition ability of the network, obtains the parameters of the network that has passed

the training, and directly applies them to the network implementation of the hardware platform, thereby avoiding the need for hardware implementation of learning circuits [9].

The MNIST handwritten digit set contains a training sample set with 60,000 sets of data and a test sample set with 10,000 sets of data. It is a subset of the MNIST database that is fully public and easy to parse, and is widely used for testing programs for machine learning or pattern recognition. The handwritten digit images in the MNIST database have been preprocessed and stored as 28 × 28 grayscale images. The data style is shown in Figure 6, which illustrates 10 × 10 data. Two metrics are generally used for evaluating the test results of the MNIST dataset, which are the recognition accuracy and the average mean square error. Recognition accuracy of the network = the number of correctly recognized images/total number of images. This equation intuitively reflects the CNN recognition ability, and the best recognition algorithms can currently achieve 99% accuracy. The recognition result of the network is expressed in the form of a vector $\{y_0, y_1 \cdots y_9\}$. Each component corresponds to the probability 0 to 9 of the recognized content, then the mean square error $\delta = \frac{1}{2}\sum(y_i - Y_i)^2$ can be used to reflect the CNN classification effect [10]. The smaller the value of the mean square error is, the closer the recognized probability is to the expected value under different classifications. In the experiments, this paper uses the average mean square error of several times as the evaluation standard, in order to avoid the extreme situation caused by a single piece of data and better reflect the overall performance of the network. In addition, in order to evaluate the operation efficiency of the network, this paper also monitors the operation time of the network, in combination with the operation power consumption of the platform, and calculates the number of images per watt per second to be recognized, in order to measure the performance-to-power ratio of the CNN operation.



Figure 6. Example of MNIST data.

3.2. Effectiveness and Analysis of PC-Based CNN Training

As described above, the CNN implemented in this paper is a five-layer network containing two convolutional layers, where the convolutional layer C1 contains six feature maps and the convolutional layer C2 contains twelve feature maps; the network maps a grayscale image with size 28 × 28 to a 10 × 1 vector. When reading the training data, the labeled values are first converted into a 10 × 1 vector so that the component representing the correct result has a value of 1 and the other components have a value of 0. This is used for error calculation during the training process [11].

In this paper, the implemented CNN is trained 60,000 times using all the data in the MNIST training sample set, and the average mean square error of this set of training data as well as the current total mean square error are outputted after every 5000 rounds of training, in order to facilitate the observation of the improvement of the recognition effect. The results of the training are shown in Figure 7. After 60,000 rounds of training, the mean square error decreases from 0.3229 to 0.0526, and the network recognition effect

is significantly improved. A total of 9239 values are correctly recognized from the final test sample set of 10,000 sets of data, i.e., the recognition accuracy reaches 92.39%. In recent years, in the handwritten digit recognition-related research literature, the recognition accuracy is generally around 95%. Although the recognition accuracy of this paper still has a certain gap with the excellent algorithms, this paper's research focuses on the FPGA-based CNN implementation program, and the current recognition accuracy can meet the basic needs of image recognition and project verification. The program running time is monitored using the time function, counted once per second, so the recognition speed is about 1.6 ms/frame during the test.

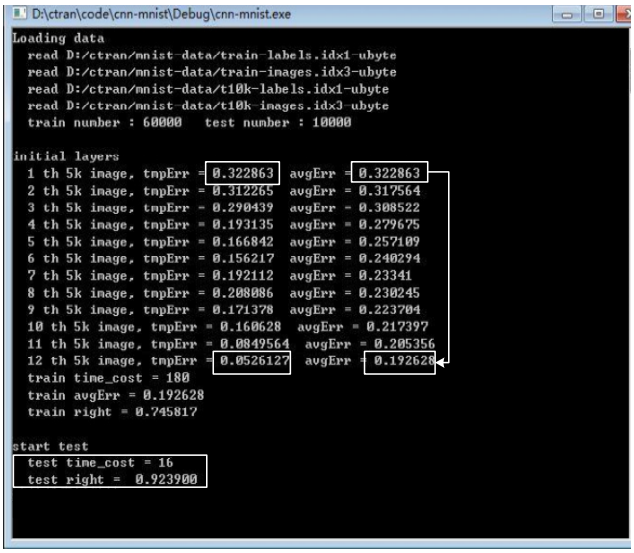


Figure 7. Training results.

In order to better observe the convergence of the network, a line chart of the mean square error is plotted based on the test data from the training results, as shown in Figure 8. From the figure, it can be seen that although there are fluctuations in the average mean square error curve every 5000 times, the overall trend is converging to zero. In response to the phenomenon of rising mean square error for the seventh and eighth datasets, this paper determines that the reason is the use of a single recognition error for weight correction, so that the correction effect is affected by individual values [12]. Furthermore, a number of data that is easily recognized incorrectly exists in the sample set. Such data can easily lead to a negative effect in weight correction, causing an increase in the mean square error. In addition, the total average mean square error curve is plotted in this paper for observing the overall convergence performance of the network. After 5000 rounds of training, the overall average mean square error of all previous training processes is calculated and plotted as a line chart. As can be seen from the figure, the overall average mean square error curve is consistently decreasing. This indicates that the data fluctuations do not affect the final convergence of the network, but only slightly slow down the convergence speed. Undeniably, the CNN implemented in this paper can still be improved in terms of recognition accuracy and convergence speed. But since this paper does not focus on the optimization of the algorithm, and the current training can already meet the demand for verifying the network accuracy and obtaining valid network parameters for FPGA-based computational acceleration testing, the related optimization issues are therefore not discussed in this paper.

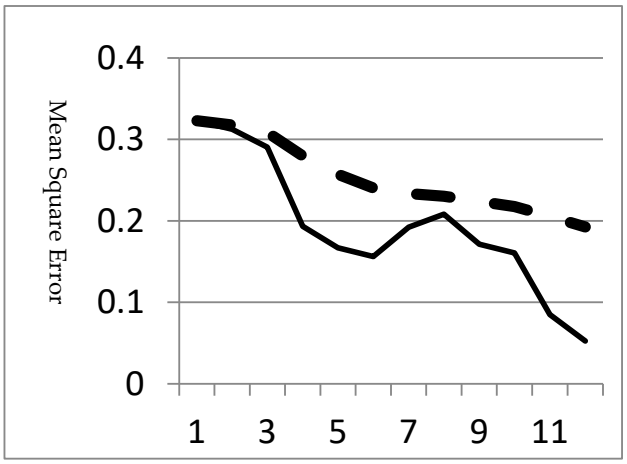


Figure 8. Mean square error curve.

3.3. Experimental Results and Analysis Based on Hardware Platform

Through the PC-based experiments, this paper obtains a set of trained experimental parameters, including the parameters of each convolutional kernel, the weight matrix of the classification layer, the bias coefficients of each layer, etc. These parameters and the test dataset are converted into a binary file, which is used to test the CNN operation effect on the hardware platform. In the experimental process, this paper realizes the program and logic configuration of the principal sample through Xilinx's JTAG simulator, and outputs the test results through the RS232 serial port. The program is run using Debug mode. At the beginning of the program, this paper uses the Dump/Restore Data File tool of Xilinx SDK 2015.4 to store the parameter file and test data file in a specific location of the DDR3 memory through the JTAG interface, which is convenient for the program to access. The address assignment and data format are shown in Table 2.

Table 2. DDR3 memory address assignment.

Starting Address	Stored Content	Data Size (Byte)
0 × 0100 0000	Weight and bias coefficient of classification layer	7720
0 × 0100 2000	Weight of convolutional layer	7800
0 × 0100 3F00	Bias coefficient of convolutional layer	144
0 × 0100 4000	Test data	7,840,000
0 × 0400 0000	Test data marker value	10,000

3.3.1. Illogical Acceleration

In this paper, the operation of the CNN ARM-based hard core of the processing unit was tested first. Because the CNN parameters have been trained to achieve better recognition results, only 10,000 sets of data from the test sample set were used in the experiment. In addition, in order to facilitate the serial output, the serial print message was set to be 10,000 times the number of correctly recognized images and the total mean square error. The test results are shown in Figure 9, the recognition accuracy is 92.39%, and the average mean square error is about 0.0717, which is basically consistent with the training results.

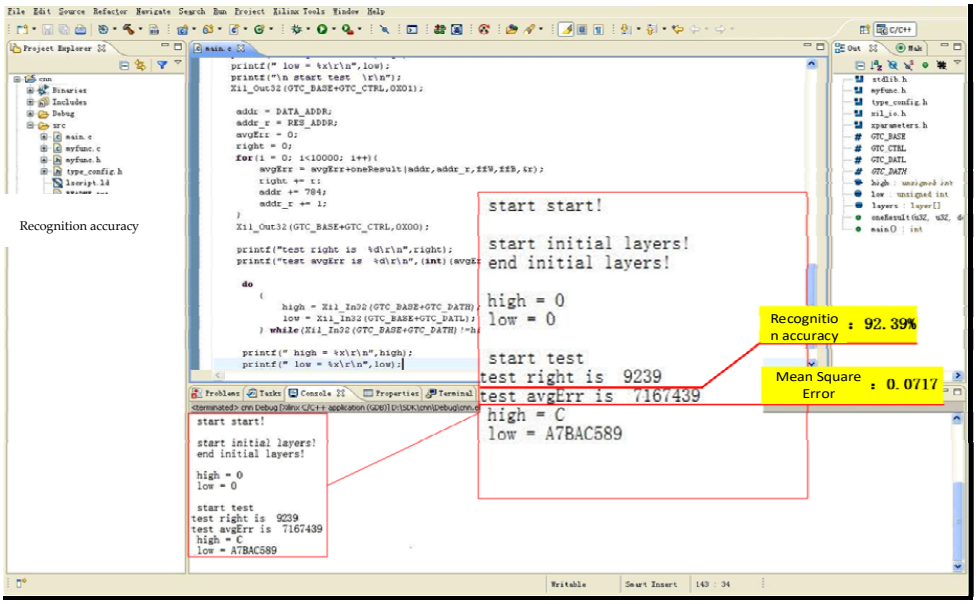


Figure 9. Hardware platform test results.

The mean square error here is slightly increased compared to the PC-based experiments, mainly due to the fact that a binary file must be used when preloading data from the PC into the DDR3 memory, and each parameter undergoes the process of being converted from a decimal to an integer and then back to a decimal again, which results in a slight loss of precision. The program runtime is monitored using the built-in global timer (GTC) of the Zynq-7000 series chips. GTC operates independently, without interference from the processing unit and programmable logic, and operates at one-half of the processing unit clock frequency, i.e., 333.333 MHz. In Figure 9, high and low are the values of the GTC's high 32-bit registers and low 32-bit registers, respectively, expressed in hexadecimal, so it can be concluded that the CNN recognition speed of the ARM-based processing unit hard core is about 16 ms/frame. There is a certain gap between this speed and the PC-based experimental results, mainly because when the hardware platform is used for experiments, only one ARM processor hard core is used, the frequency is set to 666.7 MHz, and the parallel acceleration function of the FPGA is not utilized. However, the processing unit of the PC is a dual-core I7 processor with 3.6 GHz, of which the working frequency is much higher than that of the ARM processing unit of the hardware platform. Moreover, the speed requirement of real-time image processing is generally 25 frames per second, i.e., 40 ms/frame, which is lower than the recognition speed of the current experiment.

3.3.2. Logical Acceleration

In this paper, a convolutional computation unit implemented based on FPGA is utilized to accelerate the computing of the convolutional layer. As mentioned earlier, the overall computing process of CNN is implemented by an ARM processing unit hard core for scheduling and a GTC is utilized for timing. Thanks to the flexibility of the ARM processing unit hardware core, this paper can easily accelerate the computing for a part of the program using an FPGA. In this paper, two cases of FPGA acceleration for only convolutional layer C1 and FPGA acceleration for both convolutional layers C1 and C2 are tested, respectively, and the clock frequency of the programmable logic part is 50 MHz. In this paper, 10,000 sets of test data are divided into 10 batches for testing, and the 11th set of data is the average of 10 tests. As can be seen from the figure, when only the C1 layer is accelerated, the recognition accuracy is 92.39% on average, which is the same as the result when there is no FPGA acceleration, but the recognition speed is 10.95 ms/frame, with

an improvement of about 46%. When the two convolutional layers are accelerated, the recognition accuracy is 92.4% on average, which is basically unchanged, and the recognition speed is even further improved to 4.51 ms/frame, with an improvement of about 261% compared to the case without FPGA acceleration.

When initially designing the data format, this paper refers to the experience of related studies and chooses to use 16-bit fixed-point decimals in the (1, 5, 10) format for the feature map data and 24-bit fixed-point decimals in the (1, 5, 18) format for the convolution kernel parameters. The data precision of the feature map data in fixed-point decimal format is $2^{-10} < 0.001$. The data precision of the convolution kernel parameters in fixed-point decimal format is $2^{-18} < 0.000004$. Under the current test set size, the precision loss caused by data format conversion has no effect on the recognition accuracy, which is also proved by the above experimental results.

3.3.3. Power Consumption Comparison

Finally, this paper obtains the operating power consumption of the embedded intelligent information processing hardware platform by recording the current output of the DC power supply during the experimental process, and uses the number of images recog-nized per watt per second to express the performance-to-power ratio of the hardware platform. The comparison of the above three experimental schemes is shown in Table 3. From the table, it can be seen that the computing acceleration of the convolutional layer implemented in the programmable logic part achieves good results. Although the recognition speed is still lower than the 1.6 ms/frame of the PC-based experiments when the computing acceleration is performed on both convolutional layers, the high performance-to-power ratio and the small size make the hardware platform more suitable for the airborne embedded environment.

Table 3. Performance-to-power ratio comparison.

Experimental Condition	Recognition Speed	Power Consumption	Performance-To-Power Ratio
Illogical acceleration	16.31 ms/frame	11.4 W	(5.38 frame/s)/W
C1 layer acceleration	10.95 ms/frame	11.76 W	(7.76 frame/s)/W
C1 and C2 layer acceleration	4.51 ms/frame	12.48 W	(17.74 frame/s)/W

4. Summary and Outlook

In recent years, ANNs represented by DNNs have successively made extraordinary achievements in the fields of natural language processing, image processing, speech processing, etc., which provide new solutions to meet the needs of UAVs for intelligent computing [13]. This paper firstly summarizes the typical models, learning algorithms and implementation techniques of ANNs, from which the FPGA-based implementation of CNNs is selected as the research focus. After that, this paper completes the analysis of the CNN algorithm, the design and implementation of the embedded intelligent information processing hardware platform, and verifies the feasibility of the CNN implementation in the embedded environment under the hybrid architecture of ARM processing unit hardware core + FPGA, with the Xilinx Zynq-7000 series AP SoC chip as the hardware base.

The main findings and innovations of this paper are briefly summarized below:

(1) Summarize the development history and current status of the theoretical research and implementation technology of neural networks, combine with the demand for intelligent computing in the field of aviation, choose FPGAs with the advantages of parallel computing, dynamic reconfigurability, and moderate R&D cycle as the hard-

ware basis of this paper, study and implement CNNs with outstanding performance in the field of image processing.

(2) Design a typical five-layer CNN model for the handwritten digit recognition task, complete the algorithm analysis and implementation of the forward transmission computing phase and backpropagation weight adjustment phase of this network model, use the MNIST handwritten digit set for training and testing, lay the foundation for the design of the FPGA-based implementation scheme afterwards.

(3) Design and implement the embedded intelligent information processing hardware platform for CNN computing acceleration, perform module function design for the video/image processing task in the aviation field, complete the FPGA selection and circuit design, propose the computing task division scheme for the CNN in the process of the expansion of hardware resource using multiple FPGA chips. Complete the design and optimization of data format, activation function's segmented linear approximation method based on the hardware platform, as well as the implementation of convolutional computing unit, controller and other functional modules based on Verilog hardware description language, and complete the simulation verification.

(4) Complete the connectivity test of the embedded intelligent information processing hardware platform, train the CNN implemented in this paper based on PC, use the trained network parameters, complete the CNN operation and performance evaluation on the embedded intelligent information processing hardware platform, with the average recognition accuracy of the network at more than 92%, meeting the standard of image recognition; meet the requirements of project validation, with an recognition speed of up to 4.51 ms/frame, and a performance-to-power ratio of up to (17.74 frames/s)/W.

Author Contributions: Conceptualization, Y.L. and P.S.; methodology, Y.L. and P.S.; software, Y.L. and P.S.; validation, Y.L. and P.S.; formal analysis, Y.L. and P.S.; investigation, Y.L. and P.S.; resources, Y.L. and P.S.; data curation, Y.L. and P.S.;writing—original draft preparation, Y.L.; writing—review and editing, Y.L.; visualization, P.S.; supervision, Y.L.; project administration, Y.L.; funding acquisition, P.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: This research data can be provided upon request, as the data is not publicly available due to privacy or ethical restrictions.

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

References

1. Modha, D.S.; Ananthanarayanan, R.; Esser, S.K.; Ndirango, A.; Sherbondy, A.J.; Singh, R. Cognitive Computing. *Commun. ACM* **2011**, *54*, 62–71. [CrossRef]
2. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **1943**, *52*, 99–115. [CrossRef]
3. Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **1958**, *65*, 386–408. [CrossRef] [PubMed]
4. Fang, R.; Liu, J.; Xue, Z. Design of FPGA Parallel Acceleration Scheme for Convolutional Neural Networks. *Comput. Eng. Appl.* **2015**, *51*, 32–36.
5. Merolla, P.A.; Arthur, J.V.; Alvarez-icaza, R.; Cassidy, A.S.; Sawada, J.; Akopyan, F.; Jackson, B.L.; Imam, N.; Guo, C.; Nakamura, Y.; et al. Artificial brains. A million spiling-neuron interated circuit with a scalable communication network and interface. *Science* **2014**, *345*, 668–673. [CrossRef]
6. Belk, R.; Humayun, M.; Gopaldas, A. Artificial Life. *J. Macromarketing* **2020**, *40*, 221–236. [CrossRef]

7. Kehl, W.; Jackson, M.; Fergnani, A. Natural Language Processing and Futures Studies. *World Futures Rev.* **2020**, *12*, 181–197. [CrossRef]
8. Boysen, A. Mine the Gap: Augmenting Foresight Methodologies with Data Analytics. *World Futures Rev.* **2020**, *12*, 239–248. [CrossRef]
9. Bevolo, M.; Amati, F. The Potential Role of AI in Anticipating Futures from a Design Process Perspective: From the Reflexive Description of "Design" to a Discussion of Influences by the Inclusion of AI in the Futures Research Process. *World Futures Rev.* **2020**, *12*, 198–218. [CrossRef]
10. Xu, L.; Tu, P.; Tang, Q. Contract Design for Cloud Logistics (CL) Based on Blockchain Technology (BT). *Complexity* **2020**, *32*, 143–146. [CrossRef]
11. Grant, L.; Xue, X.; Vajihi, Z.; Azuelos, A.; Rosenthal, S.; Hopkins, D.; Aroutiunian, R.; Unger, B.; Guttman, A.; Afilalo, M. LO32: Artificial intelligence to predict disposition to improve flow in the emergency department. Can. *J. Emerg. Med.* **2020**, *22*, S18–S19. [CrossRef]
12. Kirubarajan, A.; Taher, A.; Khan, S.; Masood, S. P071: Artificial intelligence in emergency medicine: A scoping review. *Can. J. Emerg. Med.* **2020**, *22*, S90. [CrossRef]
13. Grant, L.; Joo, P.; Eng, B.; Carrington, A.; Nemnom, M.; Thiruganasambandamoorthy, V. LO22: Risk-stratification of emergency department syncope by artificial intelligence using machine learning: Human, statistics or machine. *Can. J. Emerg. Med.* **2020**, *22*, S15. [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.