Application Research of CNN Accelerator Design Based on FPGA in ADAS

Jiaxiang Shi1, Xiaohua Tian2*, Zhiyuan Zheng2 and Tao Zhang2

1School of Science, University of Science and Technology, Harbin, China
2School of Science, University of Science and Technology, Harbin, China

*Corresponding author e-mail: xiaohuatian@hrbust.edu.cn

Abstract. In recent years, with the rapid development of artificial intelligence, various deep learning algorithms represented by Convolutional Neural Networks (CNN) have been widely used in various fields, showing their unique advantages. However, as we need to solve the problem, in order to obtain higher accuracy, the depth of the network is deeper and deeper, and the structure is more and more complex. In some scenarios with high real-time requirements, it is difficult to meet the application requirements. Aiming at the characteristics of a large number of multiply-accumulate operations in CNN, this paper proposes a pipelined FPGA parallel acceleration design module and studies the application in Advanced Driver Assistance System (ADAS). Altera DE1 SOC FPGA development board is our hardware implementation tool. Take 36×36 image input as an example. The running time of CNN in the FPGA at 100MHz clock frequency is 2.812ms. This time is enough for real-time Advanced Driver Assistance System (ADAS) requirements.

1. Introduction

The machine learning classification algorithm that develops rapidly around deep learning has received attention due to its superior performance. Deep learning is an area of machine learning algorithms in which computers train themselves through large amounts of data based on artificial neural networks, a form in which the human brain classifies information into machines. In particular, a convolutional neural network (CNN), one of the deep learning algorithms, is one of the most interesting classifiers in the field of image signal processing. CNN is an artificial neural network that has been developed to be suitable for data processing by mimicking human visual processing and compared to other deep learning structures, compared to other machine learning algorithms such as vector machines (SVM), decision trees, and the like, showing good performance in both video and audio signal processing.

An example of using these algorithms is the Advanced Driver Assistance System (ADAS). It can be mentioned that in ADAS, pedestrian detection (PD), vehicle detection (VD), traffic sign recognition (TSR), and forward collision warning system (FCWS), because there are many images using object recognition. The algorithm is processed, so the research of the classifier algorithm is actively carried out in the field of vehicle image processing. Specifically, the amount of training data has this richness, and recent research has shown that using CNN is better than using other classification algorithms and features such as Haar-like features or directed gradient histograms (HOG).

However, when the CNN is executed in an embedded environment such as an automotive electronic system, since the repeated convolution operation of each layer results in a long execution
In this paper, it is difficult to process in real time. In this paper, we propose a hardware architecture that improves the speed of CNN by parallel processing of repetitive operations, including convolution operations, which account for CNN real-time processing in embedded environments in CNN algorithms. The calculation amount is 86% or more. To verify the performance of the proposed hardware, we tested the vehicle detection algorithm using the Altera DE1 SOC FPGA board. The input image size used in the experiment was 36x36, and both the software and hardware environments were used for the CalTech99 data set. The detection rate was 99.69%. At 100MHz operating frequency, the hardware execution time is about 2.812ms, which can be processed in real time, which is 180 times faster than the 506.7ms measured in the software environment of the same board.

The contents of each chapter are as follows. Chapter 2 introduces the overall design ideas, including network design and hardware design. Chapter 3 introduces the experimental environment and experimental results. Finally, the conclusions of this paper are drawn.

2. Overall design

2.1. Network design

The convolutional layer is a step of generating a feature map by performing a convolution operation on the input image. The weights used for convolution operations are obtained during the learning process. In this paper, we calculated using a 5x5 convolution mask. After the convolution operation, the activation function is used for the deviation bonus. The types of activation functions include linear functions, step functions, sigmoid functions, and hyperbolic tangent functions, but the gradients change because the number of errors propagating to lower layers in the backpropagation algorithm is reduced, which is learning artificial neural networks. Always used. Almost disappeared, learning does not occur with the disappearance of the gradient. However, the introduction of a rectifying linear unit (ReLU) can solve this problem, and it is possible to learn from lower layers. In this paper, ReLU is used as an activation function because ReLU performs better than other activation functions. When performing a convolution operation, each feature map has a different convolution mask and the same convolution mask is used in the same feature map. In the "Maximum Pooling Layer", the 2×2 filter is used to select only the largest eigenvalue from the four eigenvalues of the filter, and to reduce the amount of calculation and the amount of parameters. In the final step, the fully connected layer classification is performed using the features extracted from the previous layer.

The CNN used in this paper is a CNN for vehicle detection, which is constructed by changing the LeNet-5-based network structure. In order to determine the network structure, an optimal network structure was selected by iterative experiments using multiple vehicle data. The proposed network structure consists of eight layers, including three repeating convolution layers, one maximum pooling layer and two fully connected layers. Figure 1 shows the structure of LeNet-5 and Figure 2 shows the CNN structure used in this article.

![Figure 1. Architecture of LeNet-5.](image-url)
2.2. Hardware design

This chapter describes the hardware block diagram and operation of the design. Since the convolutional layer occupies most of the execution time in the CNN structure, reducing the execution time of the convolutional layer is more efficient than reducing the execution time of other layers. Therefore, in order to reduce the execution time of the convolutional layer, the proposed hardware architecture allows convolution operations with 1 clk using 25 multipliers and continuous computation using double buffers. Figure 3 shows a complete block diagram of the designed CNN hardware accelerator, consisting of a convolution module, a maximum pool module, and a fully connected module. The operation process saves the weights, deviations and input images required for CNN operations in the block memory (weight memory, deviation memory, input image) through the bus interface, and operates sequentially from the Convolution1 module. Simultaneously perform the maximum pooling of all feature graphs, and the calculation results of each module are stored in the block memory.

Figure 4 shows a block diagram of the convolutional layer module used herein. The convolution method differs only in the size of the input buffer. Since the calculation process in the convolutional layer is the same, the first convolutional layer is briefly described. In the first convolutional layer, the size of the input buffer is 36×5, and the size of the weight buffer is 5×5 for immediate convolution calculations. The data is stored in the input buffer and weight buffer as well as the buffer size. The input buffer and weight buffer are then convolved to generate a single 32×1 feature map and another feature map to create another feature map. For each input buffer, we applied 16 5×5 convolution masks to create a line with 16 feature maps. Then we changed the input buffer and then used 16 convolution masks to create the next line of the feature map. After the operation is completed, continuous calculations In order to speed up the calculation, we use two input buffers and one weight buffer. The weight buffer size of the second and third convolutional layers is equal to 5×5, and the input buffer size is 20×6 in the second layer and 12×8 in the third layer. Create feature maps of 16×2 and 8×4 size for the input buffer.
Figure 4. Proposed Convolution Layer Module.

Figure 5 shows a block diagram of the hardware module of the largest pooling layer. To perform the maximum pooling, Figure 4 shows the four values in the function diagram. Use a large value as the result value. Since the feature map is stored in the block memory, only one value can be set to 1 clk, and the initial result appears after 6 clk. After that, the maximum merge result appears every 4 clk. The maximum pooling is performed simultaneously according to the number of feature maps. The first and second maximum pooling modules simultaneously execute 16 feature maps, and the third largest pooling module simultaneously executes 32 feature maps. Figure 6 is a block diagram of the fully connected layer hardware module. The weights and feature map values used in the fully connected layer are stored in the block memory and are calculated one by one from the memory.

Figure 5. Proposed Max Pooling Layer Module.  Figure 6. Proposed Fully Connected Layer Module.

3. Experimental environment and results

3.1. Convolution layer module

The CNN structure used herein is shown in Fig. 2. The training data set for vehicle detection contains 2128 positive images and vehicle images obtained during driving, 5130 negative images and images obtained during driving. The roads, guardrails and lanes in the image were used. The total time required to learn the data during the learning process was 1,080 seconds, and the detection rate for the 500 validation sets was 98.9%. Figure 7 shows some of the positive and negative images used in the training.

(a)positive training image
3.2. Vehicle test result

Table 1. CalTech dataset detection rate comparison

| Methods                      | Detection rate(%) |
|------------------------------|-------------------|
| PCA+SVM                      | 86.59%            |
| Gabor+SVM                    | 90.24%            |
| Wavelet+SVM                  | 90.24%            |
| Wavelet+Gabor+SVM            | 91.06%            |
| Haar-like+Cascaded AdaBoost  | 92.38%            |
| Haar-like+AdaBoost           | 94.20%            |
| Haar-like+AdaBoost+SVM       | 94.41%            |
| Proposed method              | 99.69%            |

To compare the performance of CNN and other machine learning algorithms, the vehicle detection rate was compared using the CalTech vehicle dataset. The data set used is an image of the back of CalTech's 652 car. Table 1 shows the first two chapters. The vehicle detection rate used by CNN and other machine learning methods described herein is 99.69%. It can be seen that it performs better than other machine learning methods.

3.3. FCW algorithm

Figure 8 shows a flow chart of the FCW algorithm used in the experiment. The FCW finds the lane you are driving from the input image. If there is a vehicle in the lane, after extracting the candidate group and checking whether the candidate group is a vehicle, if the collision is expected due to approaching a certain distance when the vehicle is detected, the system passes Warn the driver to prevent accidents. In the FCW algorithm, the CNN corresponding to the vehicle category is implemented by hardware, and the rest is operated by software.

3.4. Hardware verification

Figure 9 shows the Altera DE1 SOC FPGA development board as part of the embedded environment, using the DE1 SOC FPGA to verify the CNN hardware implementation. The processing system (PS) CPU uses dual-core ARM Cortex-A9, 1GB RAM, g++ 4.8.1 compiler, OpenCV 2.4.9, Linaro 14.04. The language used for hardware implementation in Programmable Logic (PL) is Verilog HDL, and the synthesizer used is Quartus Prime 17.1. The data transfer between PS and PL is performed using the AMBA Avalon bus. The hardware operates at 100 MHz and the input image used for the experiment is a 36×36 grayscale image.
Table 2. Execution time for each layer of CNN.

| Layer                  | DE1-SOC(SW) | DE1-SOC(HW) |
|------------------------|-------------|-------------|
| Avalon(write)          | -           | 0.67318     |
| Conv1                  | 71          | 0.16581     |
| Pool1                  | 1           | 0.01031     |
| Conv2                  | 287         | 0.65674     |
| Pool2                  | 0.479       | 0.00264     |
| Conv3                  | 0.221       | 0.00071     |
| Pool3                  | 0.221       | 0.00071     |
| Fully Connected1       | 3           | 0.32787     |
| Fully Connected2       | 3           | 0.00131     |
| Avalon(read)           | -           | 0.00027     |
| Total time             | 506.7       | 2.167       |

Table 2 compares the execution time of software and hardware for each layer of CNN in an embedded environment using a DE1 SOC FPGA. When implemented in hardware, the speed of each convolution layer was increased by 428 times, 437 times and 439 times, respectively. The connection layer has increased by a factor of 9. Data transfer takes approximately 0.67 milliseconds, but there is no problem with CNN hardware implemented in real time with 1.5 milliseconds.

Table 3. Execution time for each HW section

| Execution time for one image |
|-----------------------------|
| MMAP                        | 0.645         |
| Write Data                  | 0.67318       |
| CNN H/W                     | 1.49355       |
| Read Data                   | 0.00027       |
| Total                       | 2.812         |

Table 3 shows the execution time of each part of the hardware. MMAP took 0.645 milliseconds to store data in PS DDR memory, transferring data from software to hardware. The write data is the time when the data to be used in the hardware is stored in the block RAM (PL BRAM), and the read data is the time to transfer the result data of the hardware to the PS DDR, which takes 0.67318 ms and 0.00027 ms, respectively. Therefore, the hardware execution time is 2.812ms, which is the sum of the
write data, the CNN hardware execution time and the read data execution time. It takes about 355.62
fps to process an image.

4. Conclusion

Recently, the image processing method CNN using deep learning has attracted attention with its
superior performance in other machine learning methods in the field of image processing. In particular,
driver assistance systems are actively studying image processing algorithms using CNN, such as
vehicle detection, signage detection, and pedestrian detection. However, the downside of CNN is that
execution time is long because CNN has many repetitive operations. Therefore, we propose a
hardware architecture that improves the performance of CNN by processing convolution operations
and other operations that take up more than 86% of the execution time of the CNN algorithm. In the
convolutional layer, the convolution is calculated once using 25 multipliers, and a convolution
calculation is designed to display the result value per 1 clk by using the double buffer as the weight
buffer and the input buffer without interruption. In the "maximum pooling layer", the four input values
are calculated simultaneously, and the result is every 4 clk. To verify the performance of the proposed
CNN hardware, Altera's DE1 SOC FPGA development board was used and applied to the vehicle
detection algorithms used in driver assistance systems. After deploying the hardware, CNN spent
506.7ms in the embedded environment, which can process real-time at 2.812ms at 100MHz operating
frequency, which is 180 times faster than software.

References

[1] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[C]. IEEE conference on
computer vision and pattern recognition. 2015.
[2] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]. IEEE
conference on computer vision and pattern recognition. 2016.
[3] Chaudhuri K, Gentile C, Zilles S. Algorithmic Learning Theory[M]. Springer International
Publishing. 2015.
[4] WANG X H, GAO L L, SONG J K. Beyond Frame-level CNN: Saliency-Aware 3-D CNN With
LSTM for Video Action Recognition[J]. IEEE SIGNAL PROCESSING LETTERS, 2017,
24(4):510-514.
[5] CHIN CHIUNLI, WU GUEIRU. An Automated Early Ischemic Stroke Detection System using
CNN Deep Learning Algorithm[C]. International Conference on Awareness Science and
Technology, Taichung, Taiwan, 2017:368-372.
[6] LI X L, DING P L, SHI X B. Research on Bearing Fault Detection Based on Convolution
Neural Network[C]. Chinese Automation Congress, Jinan, China, 2017:5130-5134.
[7] LIU B, ZHAO W C, SUN Q Q. Study of Object Detection Based on Faster R-CNN[C]. Chinese
Automation Congress, Jinan, China, 2017:6233-6236.
[8] WENYD, ZHANG KP, LIZF. A Discriminative Feature Learning Approach for Deep Face
Recognition[C]. European Conference on Computer Vision, Amsterdam, Netherlands, 2016:499-515.
[9] AHAMMAD FAHAD, EYHALIM YAHYA. Inflectional Review of Deep Learning on Natural
Language Processing[C]. International Conference on Smart Computing and Electronic
Enterprise, Selangor, Malaysia, 2018:1-4.
[10] WANG X H, GAO L L, SONG J K. Beyond Frame-level CNN: Saliency-Aware 3-D CNN With
LSTM for Video Action Recognition[J]. IEEE SIGNAL PROCESSING LETTERS, 2017,
24(4):510-514.