Proceedings of the 2nd International Conference on Mechatronics and Smart Systems
DOI: 10.54254/2755-2721/62/20240399

Optimization research of handwritten digit recognition algorithm based on hardware neural networks

Huaijin Hu

Communication Engineering, North China Electric Power University, Bao Ding, 071003, China

220211050806@ncepu.edu.cn

Abstract. In the modern era of digitization, the widespread application of handwritten numerical data has led to an exponential increase in data volume, necessitating efficient real-time recognition and processing systems. This project is grounded in the foundational structure of neural networks, with a specific focus on integrating these networks into hardware through Very-Large-Scale Integration (VLSI). The primary objective is to achieve faster processing speeds, lower power consumption, and real-time operations. The central emphasis of this study is the design and implementation of a hardware neural network system tailored specifically for handwritten digit recognition. Drawing from neural network theory, we explore the construction of a hardware-based handwritten digit classifier. The fundamental model employed is the single-layer perceptron neuron model. Upon receiving 28x28 pixel grayscale images, the image classifier utilizes pre-trained weights, activation functions, and a maximum selector to compare and output recognition results for numerical digits. The concrete implementation is realized using the Verilog hardware description language, coupled with algorithm optimization strategies to enhance performance and efficiency. This research endeavor aims to provide an effective hardware solution for real-time handwritten digit recognition in the digital age.

Keywords: Hardware Network, Handwritten Digit Recognition, Hardware Description Language, Real-Time Recognition, Algorithm Optimization.

1. Introduction

In the modern digital age, there is an explosion of handwritten numerical data, from bank checks to educational notes, life is filled with a large amount of handwritten data. For computers to process and calculate conveniently, an efficient real-time recognition processing system is needed. Inspired by the functions of the human brain, neural networks have become an important solution to this problem. Integrating these networks into hardware through Very-Large-Scale Integration (VLSI) has always been the focus of research, aiming to achieve faster processing speed, lower power consumption, and real-time operation [1].

This project focuses on the design and implementation of a neural network system for handwritten digit recognition, specifically tailored for hardware. Based on neural network theory, the construction of a handwritten digit classifier is realized in hardware. The basic model is a single-layer perceptron neuron model. After receiving a 28x28 pixel grayscale image from the image classifier, the digit recognition result is obtained by comparing the output results of pre-trained weights, activation functions,

© 2024 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/62/20240399

and maximum value selectors. Verilog hardware description language is used for specific implementation, and optimization strategies are adopted to improve performance and efficiency. By reducing the number of adders and multipliers, efficiency is improved. This has potential application prospects in implementing high-performance digital recognition solutions in hardware design, and also promotes the development of neural networks in the hardware field.

2. Hardware Neural Networks and Handwritten Digit Recognition Algorithms

2.1. Basic Neuron Model

A single-layer perceptron serves as the fundamental unit of neural networks. The concept of perceptrons was introduced by American psychologist Frank Rosenblatt in 1957 at the Cornell Aeronautical Laboratory [2]. A perceptron consists of inputs, weights, biases, a weighted sum, and an activation function. Its operational principle involves the reception of numerical inputs, along with their corresponding weights and biases. These inputs are multiplied by their respective weights, resulting in a weighted sum, which is subsequently combined with the bias as input to an activation function, ultimately yielding an output. The schematic diagram of the single-layer perceptron neuron model is shown in Figure 1 below [3].

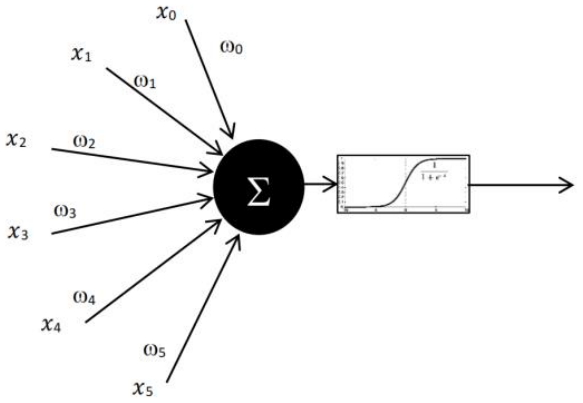


Figure 1. Schematic model of a single layer of sensory neurons [2].

In digital circuit systems, to constrain the output within a specific range, such as 0 to 1, in order to achieve the desired logical function, the utilization of an activation function is employed. This activation function transforms a given input into a specific output based on a set of predefined rules. The activation function utilized in this project are as follows in formula 1.

$$g(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

Its graphical representation is as follows in figure 2.

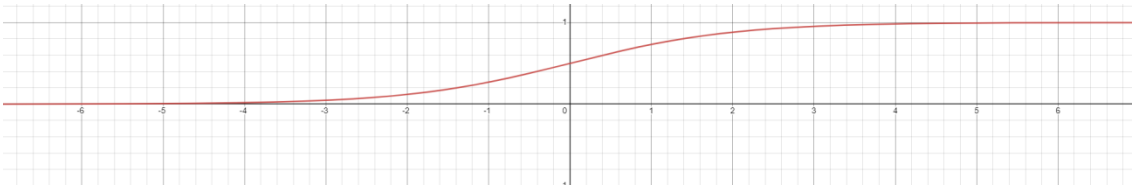


Figure 2. activation function's graphical representation (Photo/Picture credit: Original).

As observed, during the variation of the variable Z in the range of approximately -5 to 5, the value of g(z) undergoes a transition from 0 to 1, thereby achieving the logical functionality of 0 and 1 [4].

DOI: 10.54254/2755-2721/62/20240399

2.2. The principle of handwritten digit recognition algorithms

This project employs an image classifier to accomplish handwritten digit recognition,It takes in a 28x28 pixel gray scale image of a hand writing number, (white pen on a black background) recognizes it and outputs what number it is. The general workflow is as follows in figure 3.

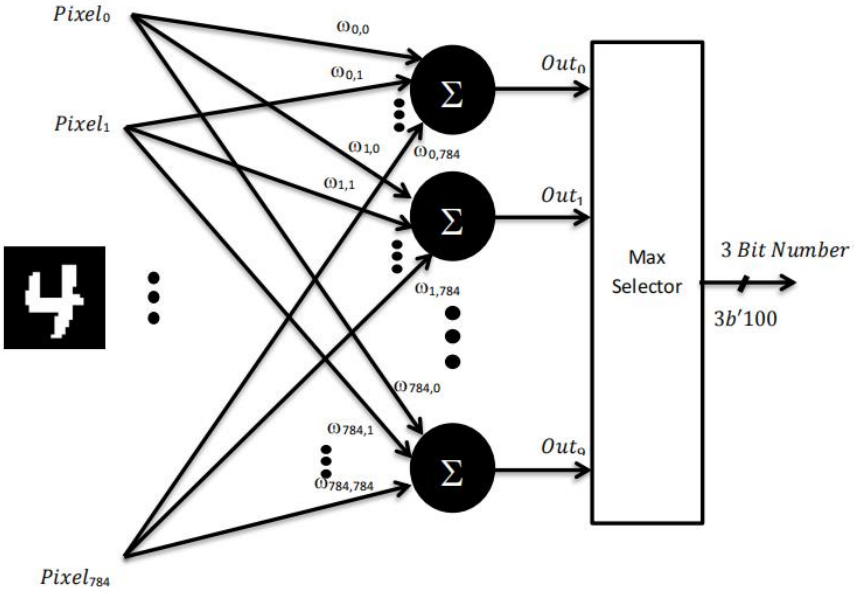


Figure 3. Digital recognition process schematic [5].

In this architecture, input pixels are derived from a 28x28 grayscale image, resulting in a total of 784 pixels. Each pixel corresponds to a grayscale value, with intensity ranging from 0 to 255. Each of these values is multiplied by a set of pre-trained weights (ω). Each neuron takes input from all pixels and produces a single output. This layer consists of ten neurons, corresponding to the ten digits from 0 to 9. The neuron outputs, to some extent, represent the probability that the input is a specific digit. Therefore, the highest output signifies the given input number. Finally, the Max Selector identifies this and outputs the corresponding digit [5].

For example, after the weight multiplication, summation, and activation function processing, the neuron's output might appear as follows: [Out0 Out1 Out2 Out3 Out4 Out5 Out6 Out7 Out8 Out9] = [0.1,0.02,0.004,0.098,0.81,0.074,0.115,0.007,0.0088,0.03]

This signifies that the network believes with the highest probability that the output is "4." The Max Selector block examines all neuron probabilities and outputs a 3-bit binary number, corresponding to the highest probability (3b'100 = 1d'4). This is the principle of recognizing handwritten digits. Since machine learning aspects are not the focus of this paper, the introduction of the machine learning part is omitted here.

2.3. Algorithmic shortcomings

If we follow the current algorithm for direct computation, the Verilog code's objective is to calculate the cumulative result for each of the 10 digits. With the direct computation approach, given that each image has 784 pixels and each pixel needs to be multiplied by the corresponding weight for each digit, we would require 10 * 784 = 7840 multipliers. Furthermore, summing up these results would require 10 * 784 - 1 = 7839 adders. However, such an approach would demand a significant number of multipliers and adders, which is not conducive to practical circuit integration. Additionally, it would consume a substantial amount of computational resources, leading to inefficient and redundant processing [6].

Hence, it is not advisable to employ a fully parallel approach. Instead, this paper adopts a pipeline and parallelization method to reduce the usage of multipliers and adders.

DOI: 10.54254/2755-2721/62/20240399

3. Optimized Design for Handwritten Digit Recognition Algorithm

3.1. Pipeline Parallelism Introduction

In the context of computational processing, pipeline parallelism is a technique that improves efficiency by dividing a complex task into a series of stages. Each stage can be executed concurrently. These stages are connected sequentially, where the output of one stage becomes the input for the next. This approach enhances resource utilization and reduces the overall processing time [7].

In the current landscape of pipeline parallelism, two prominent approaches have gained attention: Google's Gpipe and Microsoft's PipeDream, both introduced around 2019. These approaches share a similar overall design framework, but their key difference lies in how they manage gradient updates. Gpipe employs a synchronous gradient update strategy, while PipeDream follows an asynchronous approach. Notably, PipeDream's asynchronous method excels in minimizing GPU idle time.

While PipeDream's design exhibits greater intricacy, Gpipe enjoys broader popularity among a diverse audience owing to its simplicity and efficacy across a range of scenarios. Consequently, it garners wider adoption and deeper understanding within the academic community, as evidenced by prior research [8, 9]. For an illustrative example of Gpipe, please refer to Figure 4.

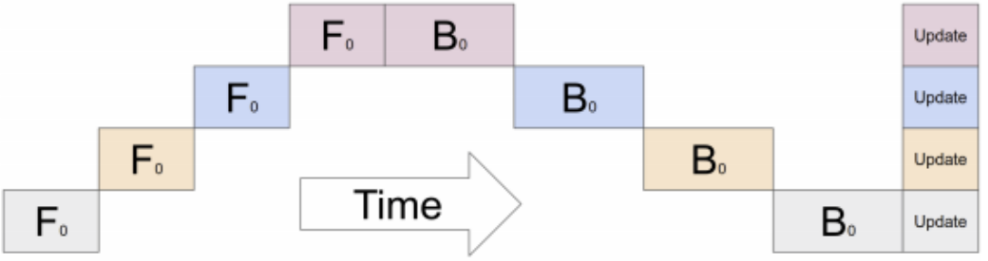


Figure 4. Schematic of the Normal Workflow [8].

In the provided diagram, the subscripts on the letters indicate batch numbers, and in this specific example, we have only one batch with subscripts all set to 0. Each row corresponds to a different GPU, and each column represents a specific timestep.

The purpose of this diagram is as follows: After GPU0 completes a forward pass, the output from its last layer is passed to GPU1 to continue the forward pass. This sequential handover continues until all four GPUs have completed their forward passes. Subsequently, the backward pass is executed one GPU at a time. After the backward pass is completed on all four GPUs, the gradients for each layer are collectively updated at the final timestep [8].

However, this design exhibits several notable drawbacks. Firstly, it suffers from Suboptimal GPU Utilization: This configuration leads to GPU underutilization, with certain GPUs remaining idle. Moreover, as the number of GPUs increases, the proportion of idling GPUs approaches 1, resulting in significant resource wastage.

Secondly, it incurs High Memory Consumption for Intermediate Results: This approach consumes a considerable amount of memory due to the storage of intermediate results. The fundamental concept underpinning pipeline parallelism extends the principles of model parallelism by introducing data parallelism. This entails dividing the original data into multiple batches and distributing them across GPUs for training. The data before division is referred to as the mini-batch, while the data subdivided at the mini-batch level is termed the micro-batch [9].

DOI: 10.54254/2755-2721/62/20240399

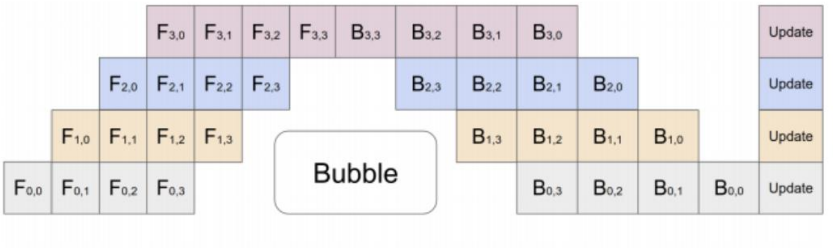


Figure 5. Schematic of the Pipeline Workflow [9].

In Diagram 5, the first subscript denotes the GPU number, while the second subscript signifies the micro-batch number. It's evident that, following the completion of training for one micro-batch, a GPU does not idle but proceeds to train the next set of micro-batches. After four timesteps, instead of processing just one batch, the GPUs have made progress on batches 0, 1, 2, and 3. This approach of segmenting batches and sequentially distributing them across GPUs resembles an assembly line or pipeline, akin to the concept of pipeline parallelism commonly employed in CPUs [7].

3.2. The principle of optimization

To reduce the number of adders and multipliers, the Verilog design adopts a hybrid approach of parallel processing and pipelining, decomposing the computation into multiple stages.

Pipelining in this context refers to the optimization of the computational process for each digit point's 784 pixels. Initially, there were 7840 multiplication operations required, one for each pixel and each digit's corresponding weight. However, by implementing pipelining for these 10 digits, only 784 multiplication units are needed. Furthermore, for each digit point, 784 multiplications are necessary. By further pipelining the pixel-wise multiplication process to calculate 98 multiplications at a time, repeated eight times, only 98 multiplication units are required. This principle effectively trades time for space, significantly reducing the number of required multiplication units. However, it's essential to note that while this reduces the number of multipliers used, it may increase the overall delay due to the pipelining process [10].

In this context, parallel processing involves the simultaneous processing of 98 data points during each clock cycle, resulting in 98 multiplication results. These 98 points are then immediately subjected to addition. The addition process follows a halving method, with additions occurring in sequences of 98-49-24-12-6-3-2-1, gradually reducing the number of points being added together. At the 49-point stage, one point is excluded from the summation. Similarly, at the 3-point stage, one point is omitted. To address this, the data from the 49-point stage is preserved and incorporated into the calculation at the 3-point stage, ensuring the accumulation of all 98 points. Finally, the sums of these eight sets of 98 points are added together to obtain the ultimate cumulative value. Figure 6 illustrates the flow of multiplier optimization in the adder optimization process.

DOI: 10.54254/2755-2721/62/20240399

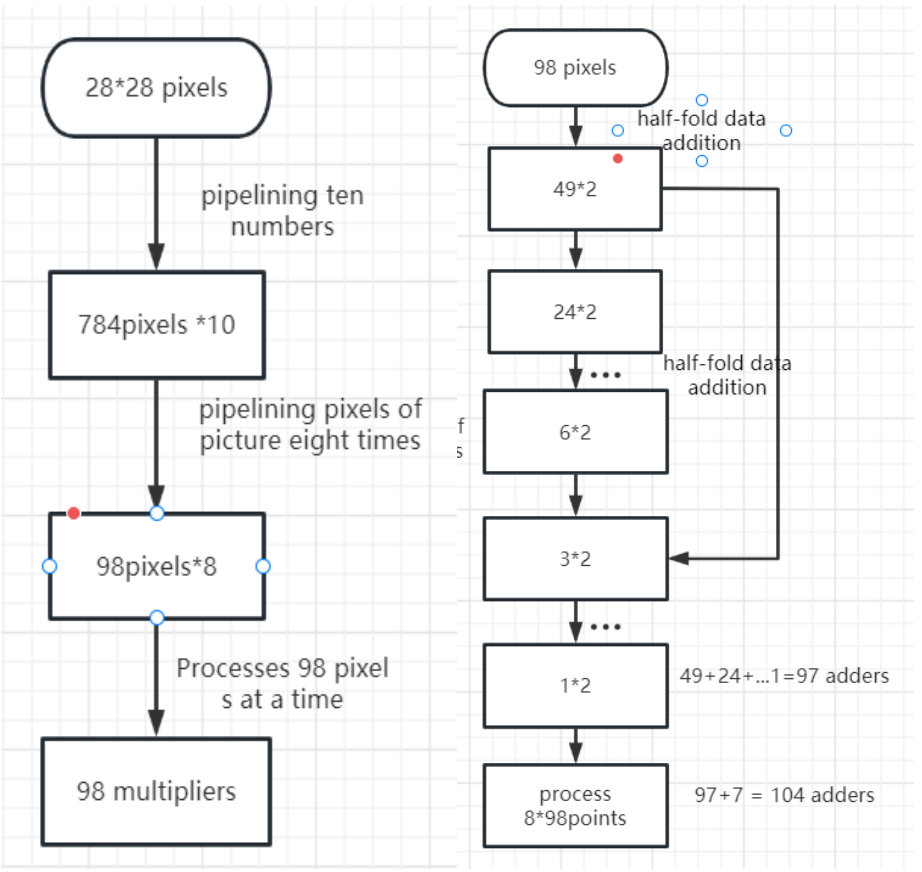


Figure 6. Multiplier optimization flow (Photo/Picture credit: Original).

3.3. The optimized result

According to the algorithm described above, the number of multipliers and adders required is as follows:

Multipliers: Simultaneously processing 98 points per clock cycle requires 98 multipliers.

Adders: For the 98 multiplication results, the following adders are needed; From 98 to 49: 98/2 = 49 adders; Since 49 is not divisible by 2, set one aside and not deal with it; From 48 to 24: 48/2 = 24 adders; From 24 to 12: 24/2 = 12 adders; From 12 to 6: 12/2 = 6 adders; From 6 to 3: 6/2 = 3 adders(The one left unused from 49 is introduced here) From 4 to 2: 2 adders; From 2 to 1: 1 adder;Total: 49 + 24 + 12 + 6 + 3 + 2 + 1 = 97 adders. Then, the sums of 8 sets of 98 points are added together, resulting in 97 + 7 = 104 adders in total.

Optimization results in the usage of only 98 multipliers and 104 adders, resulting in a delay of 115 clock cycles which is shown in table 1.

Table 1. Optimization before and after comparison.

Logical Blocks	Before	Now	Optimization
Adders	7840	104	98.67%
Multipliers	7839	98	98.74%
Power & Area	tremendous	acceptable	-

The optimization results, as shown in the above table, have significantly reduced the usage of adders and multipliers, resulting in decreased resource utilization and improved algorithm efficiency.

DOI: 10.54254/2755-2721/62/20240399

4. The testing results and analysis after algorithm optimization

4.1. Test content and results

Regarding the Verilog code that has been written, simulation tests were conducted using ModelSim. These tests were performed on ten pre-prepared images. Below are some images from the test dataset and the test results.

Table 2. Result of the test.

Test images	Actual Number	Estimated Output Number	Result
	0111	0111	correct
	0010	0010	correct
	0100	0100	correct
	1001	1001	correct
	0001	0001	correct
	0000	0000	correct

The results indicate successful recognition for all 10 datasets used in the testing.

4.2. The analysis of the test results

The analysis of the test results suggests that the algorithm and Verilog code optimization have been effective. Here are some key points to consider: Accuracy: The algorithm's ability to correctly recognize all 10 datasets demonstrates its robustness and reliability in the image recognition task. Achieving a 100% success rate is a positive outcome. Efficiency: The optimization of the Verilog code has likely contributed to the efficient execution of the recognition process. The reduced usage of adders and multipliers while maintaining accuracy is indicative of resource-efficient design. Scalability: With successful recognition on these 10 datasets, the algorithm may cloud be scalable for handling larger datasets or more complex tasks, making it suitable for various applications. Latency: While the optimization introduced a delay of 115 clock cycles, the overall reduction in computational time due to improved efficiency may offset this latency. The algorithm remains suitable for real-time or time-sensitive applications. Further Testing: It's important to continue testing the algorithm with diverse datasets and under various conditions to assess its generalization capabilities and robustness in real-world scenarios.

In summary, the test results suggest that the optimized algorithm and Verilog code are performing well, demonstrating high accuracy, efficiency, and scalability. Further testing and validation under different conditions may provide a more comprehensive evaluation of its performance.

5. Conclusion

This project aims to implement a hardware neural network for handwritten digit recognition. Through the hardware model of a single-layer perceptron neuron, the task of digit recognition has been successfully accomplished. At the same time, the existing algorithms have been optimized, effectively reducing hardware resource usage, and improving computational efficiency, demonstrating potential for practical application.

The number of test sets is still not sufficient, and larger data training sets should be used for training in the future. Similarly, in the future, the application scope of this hardware neural network can be further expanded, not only limited to digit recognition but also recognizing letters to achieve more accurate text recognition. The application scope can be extended to include image processing, embedded systems,

DOI: 10.54254/2755-2721/62/20240399

autonomous driving, and other fields. At the same time, we will continue to optimize hardware design to further reduce energy consumption and improve performance. In addition, new hardware acceleration technologies can be researched in depth to cope with more complex neural network structures and tasks.

References

[1] Jagtap V N, Mishra S K. Fast efficient artificial neural network for handwritten digit recognition. International Journal of Computer Science and Information Technologies, 2014, 5(2): 2302-2306.

[2] Bian Zhaoqi, Zhang Xuegong. Pattern recognition. 2nd edition. Beijing: Tsinghua University Press, 1999,20:1431-1438.

[3] Li Xueqiao, Ma Li. Neural Network • Engineering Application. Chongqing: Chongqing University Press, 1996:14.

[4] Huo Yuan-Lian, Zhang Qio-Sen, Zhang Jin-Shi & Fan Hong-Dong. A weighted fusion method for image de-fogging using Sigmoid function. Computer Engineering and Science 2023, (09):1621-1628.

[5] Kulkarni S R, Rajendran B. Spiking neural networks for handwritten digit recognition—Supervised learning and network optimization. Neural Networks, 2018, 103: 118-127.

[6] Zhu Hongrui, Yuan Guojun, Yao Chengji, et al. Review of distributed deep learning training networks. Computer Research and Development, 21, 58 (01) : 98-115.

[7] Narayanan D, Harlap A, Phanishayee A, et al. PipeDream: Generalized pipeline parallelism for DNN training. Proceedings of the 27th ACM Symposium on Operating Systems Principles. 2019: 1-15.

[8] Huang Y, Cheng Y, Bapna A, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. Advances in neural information processing systems, 2019, 32: 103-112.

[9] Ju Tao, Liu Shuai, Wang Zhiqiang & Li Linjuan. Deep neural network model task segmentation and parallel optimization method. Journal of Beijing University of Aeronautics and Astronautics. 2022, 48.

[10] Beohar D, Rasool A. Handwritten digit recognition of MNIST dataset using deep learning state-of-the-art artificial neural network (ANN) and convolutional neural network (CNN). 2021 International conference on emerging smart computing and informatics (ESCI). IEEE, 2021: 542-548.