electronics

MDPI

Article

Intellino: Processor for Embedded Artificial Intelligence

Young Hyun Yoon, Dong Hyun Hwang, Jun Hyeok Yang and Seung Eun Lee *

Department of Electronic Engineering, Seoul National University of Science and Technology, Seoul 01811, Korea; yoonyounghyun@seoultech.ac.kr (Y.H.Y.); hwangdonghyun@seoultech.ac.kr (D.H.H.); yangjunhyeok@seoultech.ac.kr (J.H.Y.)

* Correspondence: seung.lee@seoultech.ac.kr; Tel.: +82-2-970-9021

Received: 8 June 2020; Accepted: 16 July 2020; Published: 18 July 2020

Abstract: The development of computation technology and artificial intelligence (AI) field brings about AI to be applied to various system. In addition, the research on hardware-based AI processors leads to the minimization of AI devices. By adapting the AI device to the edge of internet of things (IoT), the system can perform AI operation promptly on the edge and reduce the workload of the system core. As the edge is influenced by the characteristics of the embedded system, implementing hardware which operates with low power in restricted resources on a processor is necessary. In this paper, we propose the intellino, a processor for embedded artificial intelligence. Intellino ensures low power operation based on optimized AI algorithms and reduces the workload of the system core through the hardware implementation of a neural network. In addition, intellino's dedicated protocol helps the embedded system to enhance the performance. We measure intellino performance, achieving over 95% accuracy, and verify our proposal with an field programmable gate array (FPGA) prototyping.

Keywords: AI processor; internet of things; machine learning; embedded system; low power

1. Introduction

As the development of computing technology enables complex computations and huge data processing, research on artificial intelligence (AI) has been promoted. Furthermore, the interest in AI computing, which includes neural networks, has also increased [1–6]. The neural network is the set of transmissions of signals between numerous calculating units and memories through entangled connections, similarly to how the brain works with spikes, neurons, and synapses [7–12]. Due to this feature, conventional computing technology with CPUs is inadequate to carry out the computation for AI [13,14]. As every neuron is sequentially computed with CPUs, the system workload is significantly increased. Even though GPUs lead the computation parallelization [15–17], the power consumption is also increased.

In order to overcome these limitations, the neuromorphic processors implemented in hardware, including neurons, were proposed [18–20]. This hardware implementation can achieve the remarkable reduction in computation time and power consumption [21–26], and realizing neuromorphic computing on the chip instead of CPUs and GPUs also helps the AI system to become smaller. These advantages lead the hardware realized computation module for the AI system to be applied to not only in high-performance servers but also edge devices of the internet of things (IoT). By including the AI computing in the edge, the IoT can swiftly perform the AI operation and significantly reduce the workload of the system core on the edge. For the AI edge as an embedded system, it is essential that the computation module should be realized with restricted resources as well as operated with low

power [27–33]. Therefore, analyzing the algorithms and neurons which occupy the resources of the AI processor and applying the processor to embedded AI system according to applications are important.

In this paper, we present the low-power processor with less resource usage, named intellino, for embedded AI systems. Intellino operates in low power with fully excluding multiplications thanks to the optimized AI algorithms. Parallelized neurons which form the intellino's neural network-based architecture help the system core to reduce the workload. Moreover, the designated protocol for intellino enables the embedded system to work efficiently through the system core. We evaluate intellino's performance with a simulator based on the optimized algorithms. We also analyze the workload of the AI system and the intellino requirements for integrated systems by implementing intellino on an field programmable gate array (FPGA). Finally, we demonstrate our proposal's feasibility through prototyping.

The rest of this paper is organized as follows. Section 2 describes the AI algorithms functionally and presents the algorithms optimizing for hardware realization. Section 3 explains the details of the intellino microarchitecture and presents the optimization for an embedded system. Section 4 provides the measurement result of intellino performance with the simulator based on the optimized algorithms. Section 5 describes the workload of the AI system and the requirements with intellino through specific AI applications. Section 6 demonstrates the realization and experimental results of intellino with FPGA prototyping. Section 7 summarizes our proposal and presents the future work.

2. Artificial Intelligence Algorithm

As each neuron in neural network includes an AI algorithm and a number of neurons are built in the AI system, the AI algorithm directly affects the system characteristics. In case of hardware implementation, the considerable neurons occupy most of the resources. Therefore, adopting the proper algorithms and realizing the optimized algorithms are important. For intellino, we adopt two AI algorithms, *k*-nearest neighbor (*k*-NN) and radial basis function neural network (RBF-NN). The algorithms are optimized for hardware realization.

2.1. k-NN and RBF-NN

The *k*-NN is one of machine learning algorithms for classification or regression, based on the distance computation. The *k*-NN classifies a test dataset by comparing distance between the test dataset and training dataset. For the supervised learning, the training dataset should be learned before classification, and each training dataset has its own category and location information (*vector*) for the distance calculation. As the test dataset also has its own *vector*, the algorithm performs classification with calculating the distance with each training dataset's *vector* and test dataset's *vector*. After taking the test dataset, the algorithm has the number of distance results, which equals to the number of trained datasets. The algorithm takes *k* variable and classifies the test dataset as the major category among the nearest distance of *k* trained dataset. When the k value is 1, the test dataset is classified as the nearest trained dataset. When the k value is 3, the test dataset finds the three nearest trained datasets and is classified as the major category among the sorted dataset. The *k*-NN can prevent the under-fitting or over-fitting issue through the *k* variable and perform the classification with only distance calculation.

The RBF-NN is a single layer (input, one hidden, output) neural network, which includes Gaussian distribution as an activation function in the hidden layer. When the input layer gets a training dataset, the hidden layer builds (trains) probability distributions depending on the previous training dataset. When the training datasets which have the same category are close, the hidden layer builds a narrow and high probability distribution. Otherwise, with scattered training datasets, the distribution is formed as wide and low. When another category of training dataset is learned and located on the built distribution of different category, the distribution becomes small to minimize the interference with other distributions. When the input layer receives a test dataset, the output layer outputs the probability values according to all categories of distributions. Thanks to the flexible distribution of each category, the RBF-NN can perform a precise classification.

2.2. Optimization for Hardware Realization

Intellino performs AI algorithm-based computation in the numerous neurons. As the computation influences the power consumption and resource usage of intellino, the algorithm optimization for hardware realization is important. In order to realize two algorithms in the hardware, we simplify the arithmetic operation and utilize proper properties of the algorithms. The multiplication is fully excluded by applying Manhattan distance calculation instead of Euclidean distance. The adder unit consumes less power and resources than the multiplication unit. In case of 8-bit calculation, a multiplier uses about 10× more resources than an adder. However, the accuracy only grows by about 1% with the multiplier compared with the adder. The adder is enough to fulfill the algorithms performance with less power consumption. Additionally, the algorithm computation units can be retained by each neuron without sharing and reusing, which enables the parallel computation. In addition, the probability of RBF-NN is substituted to matching/non-matching by setting a threshold on the distribution. Whenever the training dataset is learned, each training dataset has an influence field (IF) which decides the test dataset to be classified as its category or not. Unlike the probability distribution, the IF of matching/non-matching is generated by only distance calculation, and this leads the reduction in calculations.

According to the AI algorithm optimization, the main difference between *k*-NN and RBF-NN is the categorization method. While the *k*-NN recognizes any test dataset to only the nearest trained dataset's category, the RBF-NN recognizes some test datasets to additional categories (*uncertainty* and *unknown*). In the case of RBF-NN, each trained dataset has the IF by threshold. A test dataset is generally recognized to the nearest category of trained dataset. When the test dataset is located on the IFs intersection of two or more other categories, the test dataset is recognized as *uncertainty*, and when the test dataset is not involved in any IFs, the test dataset is classified as *unknown*. The optimized algorithms are realized on intellino, and this makes intellino operate with low power with less resource usage.

3. Intellino Architecture

Intellino is the AI processor involving optimized features for embedded systems. The AI algorithms described in Section 2 are included in the neurons, and this leads to the low power consumption and lower resource utilization of the system. The single neural network-based architecture with parallelized neurons reduces the workload of the system core by facilitating the simultaneous massive recognition. The dedicated protocol enhances the repetitive AI operation (learning/recognition) through the *interface* module. The microarchitecture of intellino with operation flow and the details of hardware realization to apply intellino in an embedded system are described below.

3.1. Microarchitecture

Figure 1 shows the block diagram of intellino. Intellino includes an *interface* for communication with system core and an *operator* for learning/recognition with *neuron cells* (*N-cells*). The system core receives an external signal for learning/recognition and transmits a training/test dataset to the *interface*. A *data transceiver* receives the data from the system core, which implies the dataset or command for learning/recognition. A *finite state machine* (*FSM*) interprets the consecutive data which are organized by the designated protocol. An *instruction encoder* encodes the interpreted data and announces to the *operator*.

The *operator* performs the operation of learning/recognition with the transmitted data. The *instruction decoder* receives the data/category/distance of the dataset, the writing/reading signals, and the selection of algorithms. When the signal indicates writing data and category, the *operator* learns the training dataset. When the signal indicates writing data and reading category or distance, the *operator* recognizes the test dataset. As the *instruction decoder* informs the completion signal of the *operator* to the *instruction encoder*, the *instruction encoder* can confirm whether the *operator* finishes the instruction. Additionally, the *instruction encoder* transmits the algorithm signal and the *operator* classifies the test dataset with the delivered algorithm signal.

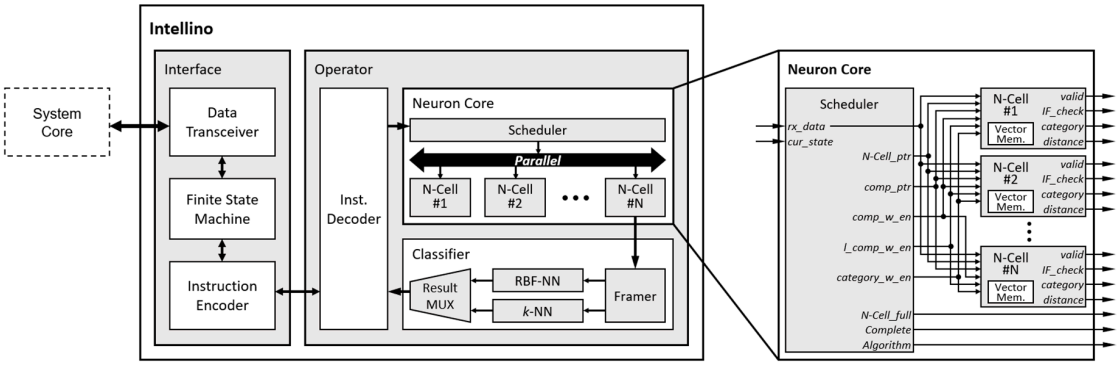


Figure 1. Block diagram of intellino and *neuron core* module details.

The decoded data make the *neuron core* to operate learning/recognition. The *neuron core* consists of a *scheduler* and *N-cells* in parallel connection for storing the training dataset in a *vector* memory and calculating the distance of test dataset. The *instruction decoder* informs a *current state* (*cur_state*) and *received data* (*rx_data*) to the *scheduler* according to the instruction. In the case of the instructions for the learning, the *scheduler* sequentially receives the training dataset and increases a *component pointer* (*comp_ptr*) which indicates the index of *received data* at the dataset. The *received data* are delivered to the *N-cell*, which is assigned by a *N-cell pointer* (*N-Cell_ptr*), and stored in the *vector* memory at the *component pointer* pointed. After all the data of training dataset are stored, the category value is delivered with a *category write enable* (*category_w_en*) signal. By increasing the *N-cell pointer*, the next *N-cell* gets ready for the next learning. In the case of the instruction for the recognition, the *scheduler* also increases the *component pointer* whenever the data of test dataset are received. In every trained *N-cell*, the stored (trained) data, which are pointed by the *component pointer* in the *vector* memory, are calculated with the received test data. The calculated value is accumulated for the total distance between the trained dataset and test dataset. When the *scheduler* notifies the last data of test dataset with a *last component write enable* (*l_comp_w_en*) signal, every trained *N-cell* finishes the calculation and outputs the accumulated distance and trained category with a *valid* and an *IF check* signals. The *scheduler* also informs the algorithm to be classified with to the *classifier*.

The calculation results of distance and category of each trained *N-cell* are combined through the *framer* in the *classifier*. The classification with the framed *N-cells* data is performed in *k-NN* or *RBF-NN* module according to the selected algorithm. The *k-NN* and *RBF-NN* module of *classifier* find the minimum distance to classify the test dataset as the nearest trained dataset. Thanks to the hardware realization, the searching minimum is parallelly conducted based on the divide and conquer methods. Every distance of trained *N-cells* is compared in pairs; for instance, the minimum searching only takes 10 clocks against the 1024 trained dataset. Especially in the case of *RBF-NN*, the test dataset is classified as *uncertainty* or *unknown* by the *IF check* signals. As each of the framed *N-cells* data have the *IF check* signals, the *uncertainty* or *unknown* is determined by the category identifying as well as the divide and conquer methods. When the compared pair have different categories with *IF check* signal '1', the *IF check* signal is maintained, and the result indicates the *uncertainty*. When the pair have *IF check* signal '0', the result indicates the *unknown*. Finally, when the system core requires the recognition results, the *instruction encoder* transmits the reading category or distance signal to the *instruction decoder*. The recognized category and distance results are delivered through the *interface* according to the reading signals.

3.2. Optimization for Embedded System

In order to reduce the workload of system core, the calculation of *N-cell* operates in parallel. For the parallelization, each *N-cell* includes the *vector* memory, which is the assembly of *components*. The dataset matches the *vector*, and each datum of the dataset is related to the *component*. In the process of learning, the input data are sequentially stacked up in the *components* of the *N-cell*. The category is

also allocated to the *N-cell* and the next *N-cell* gets ready for the next dataset learning. In the process of recognition, the input data are calculated with every *component* of the trained *N-cell*. Whenever the data are received, every trained *N-cell* calculates and accumulates the distances, simultaneously. Thanks to this feature, the recognition time is not increased, even if the number of trained *N-cell* gets increased.

The number of *N-cell*s and the size of the *vector* not only decide the usage of hardware resources but also affect the system accuracy. The increment of the *N-cell*s enables the system to train more dataset, and it is suitable for the application that has coarse-grained dataset with numerous categories. On the other hand, the increase in *vector* size enables the system to distinguish in detail, and a large *vector* size is appropriate for the application that has a fine-grained dataset. Therefore, the *N-cell* counts and *vector* size of intellino can be decided depending on the application of embedded system.

Intellino also facilitates the efficient learning/recognition through the dedicated protocol. As intellino targets an embedded system, the *data transceiver* employs the serial interface to reduce pin mapping with the system core. Also, the *FSM* enables the repetitive data receiving for successive learning/recognition. As shown in Figure 2, the *interface* communicates data in 8-bit. The first 8-bit is a *setup value*, which involves the information of single/sequential mode, write/read mode, and an *address* for the *neuron core* operation. When the single mode, the 16-bit data are attached right after the *setup value* for receiving the read data or transmitting the write data. In the case of sequential mode, the dataset length in 16-bit is notated, and the consecutive data, which include *components* and category, are delivered. For example, the learning is executed by *write category* (*CAT*) command, transmitting the *setup value*, data length, and dataset with category. The recognition is performed with *write last component* (*LCOMP*) command, transmitting the *setup value*, data length, and dataset. Intellino prints out the result of recognition by *read one* command, which shows the classified category or the calculated nearest distance depending on the *address*. The *address* indicates the command for data/distance/category or algorithm and enables the system to get the desired result with the *read one* command. As the interface is full duplex, the *read one* command transmits dummy data for receiving the desired data. Through this method, the protocol enables the system to perform learning/recognition without additional overhead, even though the dataset is bigger. Furthermore, since the protocol is based on the serial interface, the system can generate a learning/recognition library with a built-in interface and operate intellino efficiently.

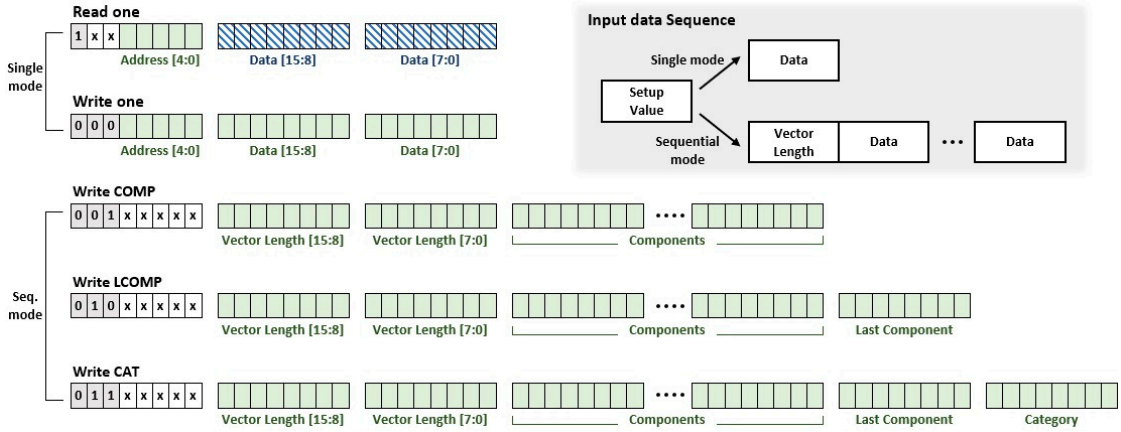


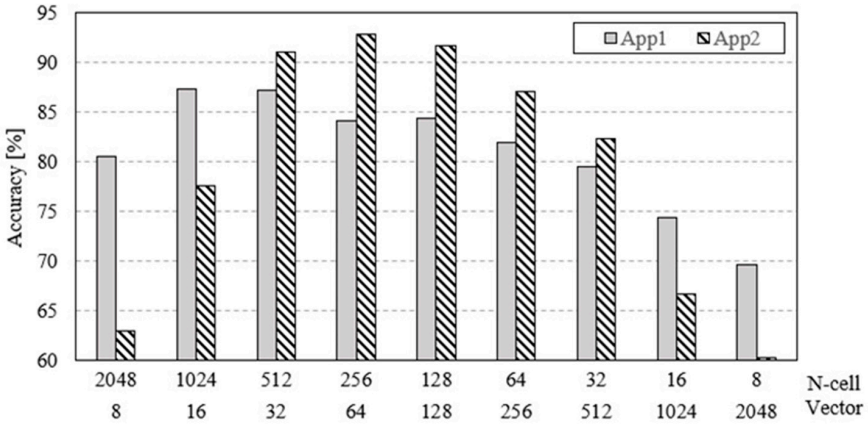
Figure 2. Protocol for learning/recognition of intellino.

4. Simulator

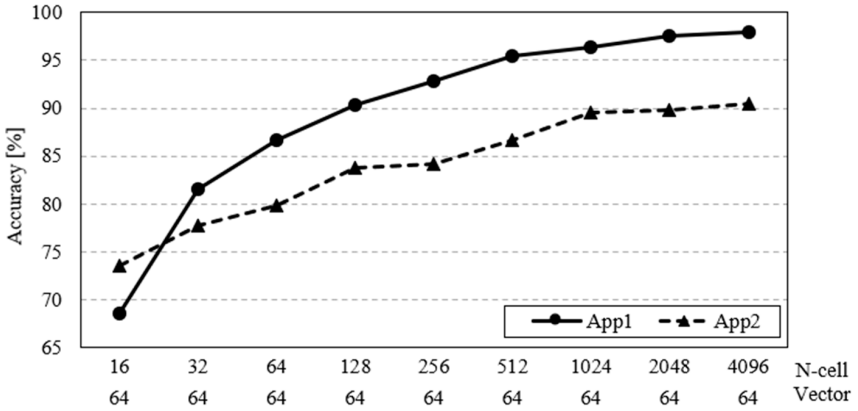
In order to evaluate the performance of intellino, we built a simulator including the hardware optimized algorithms. For the same operation as intellino, the simulator has the same structure by including the *neuron core* and the *N-cells*. Each *N-cell* retains the *vector* and optimized algorithms for learning/recognition and is contained in the *neuron core*. For the evaluation of intellino in accordance

with various datasets, the *neuron core* can configure the number of *N-cells* and the size of the *vector*. Through the constructed simulator, we measure the accuracy of intellino.

Figure 3 shows the relation between the *N-cell* counts, *vector* sizes, and accuracy depending on the applications. In Figure 3a, application 1 (App1) includes audio-based dataset [34] with two categories, the application 2 (App2) includes an image-based dataset [35] with five categories. As raw data are inadequate for learning/recognition in terms of accuracy as well as data quantity, the raw data should be pre-processed and organized for extracting the feature data for the efficient AI operation. In the case of App1, the audio-based data are pre-processed by a short time Fourier transform (STFT) [36], which extracts the feature regarding the frequency over time. As App1 is for the classifying words and each word has its own frequency, the extracted frequency dataset is used for the recognition. In the case of App2, which is for classifying hand-written digit images, as the image-based dataset is well-organized and the feature dataset by itself, the resizing with linear interpolation is only performed according to the *vector* sizes. In order to perform learning/recognition with the neurons of intellino, the 2D featured datasets of audio and image are flattened into a 1D array. As shown in the graph, App1 shows the highest accuracy of 87.4%, with 1024 *N-cells* and 16 *vector* sizes, and App2 has an accuracy of 92.8% with 256 *N-cells* and 64 *vector* sizes. The experimental results show that each application has the suitable combination of the *N-cell* counts and the *vector* sizes.



(**a**)



(**b**)

Figure 3. (**a**) Intellino accuracy depending on *N-cell* counts and *vector* sizes; (**b**) Intellino accuracy with fixed *vector* size.

We also analyze the correlation between the number of *N-cell* and the intellino accuracy. Figure 3b describes the accuracy with the App1 and App2 datasets when the *vector* size is fixed at 64 bytes, and the number of *N-cell* is increasing. App1 and App2 show a similar trend, which indicates the

growth of accuracy with the increment of *N-cell* counts. Especially for App2, when the number of *N-cell* is increased from 16 to 32, the accuracy grows from 68.6% to 81.6%, which shows the 13% increment of accuracy. Otherwise, when the number of *N-cell* is increased from 2048 to 4096, the accuracy only grows by 0.4%, from 97.6% to 98%. The accuracy of App2 converges around 97%, even if the count increases. In addition, the accuracy of App1 has a slight increment with the enormous *N-cell* around 90%. Therefore, the *N-cell* counts and the *vector* sizes should be determined in accordance with the target applications and the embedded system environments for the accuracy and resource usage. Thanks to the reconfigurable feature of the FPGA, intellino can be implemented with appropriate neuron specification.

5. Workload Analysis

A recognition application with AI is conducted through the series of processes. In the case of learning, the pre-processing raw data and the storing of training dataset should be carried out. For the recognition with AI system, the pre-processing, calculating for classification, and informing of the recognized result should be carried out. The pre-processing is performed for generating the training and test dataset. As the training dataset is prepared in advance and the test dataset is delivered one by one, parallel computing for prompt pre-processing is not essential. The system core in the embedded system is enough to perform the pre-processing. The storing and the calculating are relevant on account of sharing the dataset memory. The storing is sequentially proceeded, since the training dataset is transmitted in sequence. Otherwise, the calculating for classification is executed as many times as the trained dataset. As the test dataset is compared with every trained dataset, a greater number of trained datasets leads to more calculation. The recurring recognition with the system core can cause a bottleneck issue due to the drastic increment of calculation. The parallelization of calculating in intellino reduces the workload of the system core and enables the embedded AI system to process the test data without delay. After the classification, the result of recognition is informed to the system core, and the result is directly displayed or utilized according to the applications.

In order to analyze the concrete workload of AI system, we implement intellino on the FPGA, as shown in Figure 4. The *neuron core* with 32 *N-cells* and 256 *vector* size is implemented on intel DE2-115 evaluation board with Cyclone-IV FPGA. We adopt a face expression recognition application to demonstrate the real-world system applicability of intellino. We generate the training dataset for classifying the face expression of three categories, smile/angry/snooze, and pre-process the raw data and extract the feature data with a CPU-based system. The raw face image is taken from the camera, and the face region is cropped with Haar-like feature [37] for excluding the background. The cropped image is converted into a grayscale image, and Canny edge detection [38] derives an edge for acquiring the feature of face expression. For the more efficient recognition of expression with the face image, the regions near to the eyes, nose, and mouse are extracted for generating the feature dataset. After the pre-processing, the featured training datasets are stored (trained) in each of *vector* memories of intellino. With trained intellino, the recognition of face expression is performed. The image for recognition is also taken from the camera and pre-processed with the same process as the training dataset. The pre-processed test dataset is delivered to intellino and calculated with every trained dataset in intellino. Thanks to the parallelized neurons, the multiple calculations for classification are promptly conducted. Even when the trained dataset is increased in intellino, the calculation time for recognition is always the same. Then, the result of recognition is transmitted to the system and displayed. The recognition achieves 90.19% accuracy with 36 datasets by *k*-fold cross validation, 12 datasets for each category and 10/2 for training/test datasets, and the operation with real-time capturing demonstrates the functionality of intellino. Through the face expression recognition application with the classical camera, we analyze the workload of the AI system step by step. Although the AI system has the most workload at the recognition step, the parallel computation of intellino takes charge of the repetitive recognition and improves the system performance by lessening the workload. As intellino receives the feature data for learning/recognition, the system can substitute other sensors

for the classical camera. Finally, as the pre-processing in accordance with the attached sensor generates and delivers the feature data to intellino, we also discover that intellino only requires the interface with the *neuron core* for the data transmission with the system core, which lightens the peripherals of intellino.

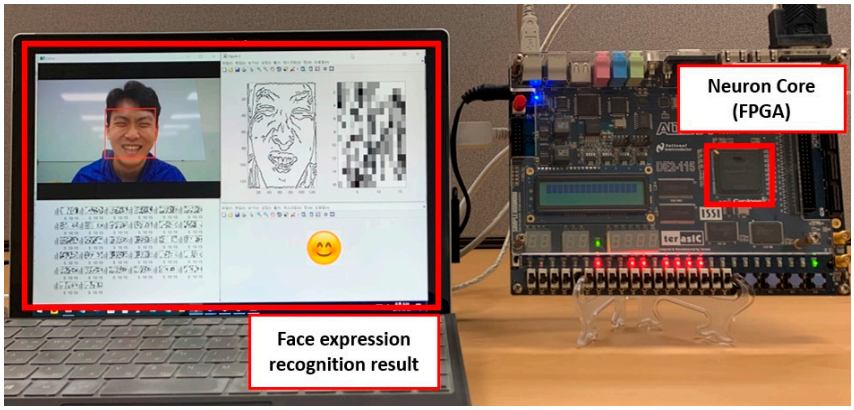
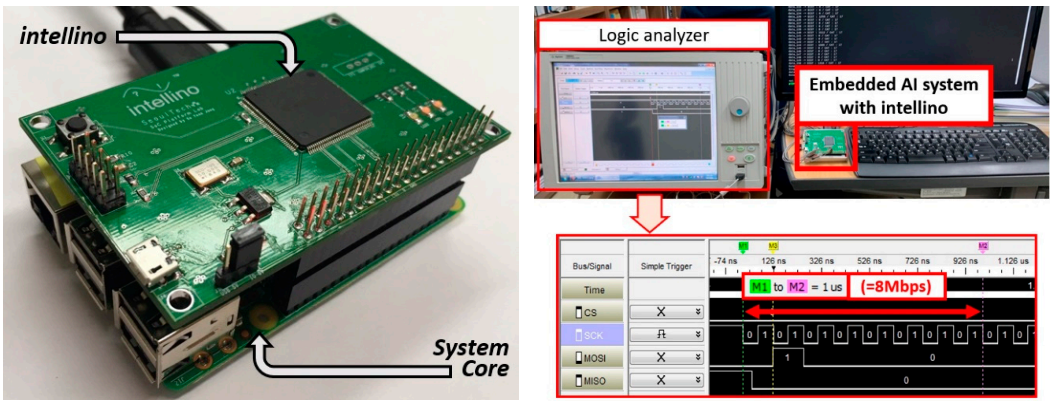


Figure 4. Implementation of *neuron core* on FPGA.

6. Realization

After the workload analysis, we finally design the prototype board of intellino for an embedded AI system. Figure 5a shows the integrated AI system with intellino, which is realized on intel MAX10 FPGA and supports up to 88 MHz operation. The prototype board includes the FPGA and joint test action group (JTAG) connector for intellino realization and the general-purpose input/output (GPIO) header for integration. The header enables the system core to be easily connected to intellino and to get external signals for learning/recognition. As intellino adopts a serial interface while employing fewer pins, which is directly related to the size of edge device, the system core can utilize the other GPIOs for various applications. Intellino achieves the same accuracy as the optimized algorithm-based simulator and performs according to the analyzed workload. Therefore, we can operate various AI applications based on image and audio by building the embedded AI system with prototyped intellino. In order to perform the AI operation with the embedded system, we develop a library for the system core including the learning/recognition functions. As the library is based on the built-in interface of the system core and the designated protocol of intellino, the embedded system can easily perform the repetitive AI operation of learning/recognition with pre-processed datasets.



(**a**) (**b**)

Figure 5. (**a**) Integrated AI system with prototyped intellino; (**b**) experimental result of intellino.

With the prototype board and the built library, we also analyze the performance of the proposed intellino in terms of operation time. We implement intellino with the *neuron core* of 16 *N-cells* and 1024

vector size, which also indicates the applicability of diverse specification, and measure the performance with the face expression recognition application. We increase the *vector* size of intellino and generate the pre-processed face expression dataset with more featured data than the experiment in Section 5 to analyze the operation time with a larger dataset. The training/test datasets are delivered to intellino for learning/recognition, and the communication between intellino and the system core is conducted with a serial peripheral interface (SPI) protocol. A serial clock (SCK) in the SPI communication determines the transmission speed, and we set the SCK frequency to 8 MHz, which enables the 8 Mbps communication, as shown in Figure 5b. Through this communication, intellino performs the recognition of one test dataset (1024-byte) in about 1ms at 50 MHz operation. Even if the trained dataset is increased, the recognition time is maintained thanks to the simultaneous calculating with parallelized neurons. This means that only the number of test dataset recognition determines the system operation time, not the number of trained datasets. Thus, the integrated embedded system takes 30 ms to recognize 30 test datasets, regardless of trained datasets. In order to process 30 datasets (frames) per second for real-time performance, intellino requires just 1.5 MHz operating frequency, which leads to the reduction in power consumption for the computation, rather than higher operating frequency. Consequentially, the experimental result demonstrates the feasibility of our proposal and real-time operation for embedded AI system with intellino.

7. Conclusions

In this paper, we presented the low-power processor, intellino, for embedded AI system. For intellino, we analyzed and optimized the distance-based AI algorithms for hardware realization. By fully excluding multiplication, the optimized AI algorithms enable intellino to operate under low power and exploit fewer resources. We also implemented the parallelized neurons as well as the optimized algorithms. By simultaneously computing in the neurons, the AI operation is performed shortly, and the system core reduces its workload. Furthermore, the designated protocol for repetitive learning/recognition improves the system performance. Thanks to these hardware designs, intellino can be applied to the embedded AI system, including the edge of IoT, while the multicore with deep neural network (DNN)- or spiking neural network (SNN)-based processors [18,20,26,39] becomes the solution of complex AI system. Intellino can conduct the classification for face expression, as described in Section 5, the recognition of driver condition or speaker with smart navigation, the inspection for surface mount technology (SMT), the notification of parking lot space, and other tiny AI applications. Thanks to the architecture of intellino and the feature of FPGA, intellino also has flexibility in specification. While other AI processors targeting the embedded system, such as Intel Curie with 128 *N-cells* and 128 *vector* sizes or General Vision NM500 with 576 *N-cells* and 256 *vector* sizes, have a fixed specification, the reconfigurability of intellino enables the system to be set with the optimized specification. Through the simulator based on the optimized algorithms, we evaluated intellino, which achieves over 95% accuracy. In addition, we analyzed the detailed workload of a system with intellino through the AI application. Finally, we verified intellino, which supports up to 88 MHz operation, with FPGA prototyping and demonstrated the feasibility of our proposal.

In future work, we will compare the performance of intellino with various AI algorithms and neural networks in aspect of the resource usage and the accuracy. A support vector machine (SVM) for detail classification, principal component analysis (PCA) for unsupervised learning, a DNN or SNN for complex application, and other algorithms and networks can be adopted for the advanced intellino. As the computation units for the algorithms and neural networks have an effect on the AI system, the computing optimization method and network communication will be considered. We will also integrate the hardware realized pre-processing, such as edge detection, STFT or object movement detection [40], with intellino to optimize for specific applications. Furthermore, we will apply the multiple intellino for a multicore AI system. Since the multicore AI system needs a specific module for controlling the cores [41], we will take the efficient multicore management into account. We expect that

the multi-intellino will enhance the performance of embedded AI systems and the flexibility against diverse applications.

Author Contributions: Conceptualization, Y.H.Y.; data curation, D.H.H.; investigation, Y.H.Y. and D.H.H.; methodology, Y.H.Y. and J.H.Y.; validation, D.H.H. and J.H.Y.; writing—original draft, Y.H.Y.; writing—review & editing, Y.H.Y. and S.E.L.; supervision, S.E.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea) under Industrial Technology Innovation Program. No. 20006566, 'Development of embedded artificial intelligence module and system based on neuromorphic'. This research was also funded by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT). No. 2019R1F1A1060044, 'Multi-core Hardware Accelerator for High-Performance Computing (HPC)'.

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

References

1. Choi, H.S.; Park, Y.J.; Lee, J.H.; Kim, Y. 3-D Synapse Array Architecture Based on Charge-Trap Flash Memory for Neuromorphic Application. *Electronics* **2020**, *9*, 57. [CrossRef]
2. Sun, S.Y.; Li, J.; Li, Z.W.; Liu, H.S.; Liu, H.J.; Li, Q.J. Quaternary synapses network for memristor-based spiking convolutional neural networks. *IEICE Electron. Express* **2019**, *16*, 1–6. [CrossRef]
3. Amirhossein, T.; Anthony, S.M. A spiking network that learns to extract spike signatures from speech signals. *Elsevier Neurocomput.* **2017**, *240*, 191–199. [CrossRef]
4. Cai, W.; Hu, D. QRS Complex Detection Using Novel Deep Learning Neural Networks. *IEEE Access* **2020**, *8*, 97082–97089. [CrossRef]
5. Dang, H.; Liang, Y.; Wei, L.; Li, C.; Dang, S. Artificial Neural Network Design for Enabling Relay Selection by Supervised Machine Learning. In Proceedings of the 2018 Eighth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC), Harbin, China, 19–21 July 2018; pp. 1464–1468. [CrossRef]
6. Shen, G.; Yuan, Y. On Theoretical Analysis of Single Hidden Layer Feedforward Neural Networks with Relu Activations. In Proceedings of the 2019 34rd Youth Academic Annual Conference of Chinese Association of Automation (YAC), Jinzhou, China, 6–8 June 2019; pp. 706–709. [CrossRef]
7. Moon, S.; Shin, J.; Shin, C. Understanding of Polarization-Induced Threshold Voltage Shift in Ferroelectric-Gated Field Effect Transistor for Neuromorphic Applications. *Electronics* **2020**, *9*, 704. [CrossRef]
8. Adarsha, B.; Anup, D.; Yuefeng, W.; Khanh, H.; Francesco, G.D.; Giacomo, I.; Jeffrey, L.K.; Nikil, D.D.; Siebren, S.; Francky, C. Mapping Spiking Neural Networks to Neuromorphic Hardware. *IEEE Trans. VLSI* **2020**, *28*, 76–86. [CrossRef]
9. Mantas, M.; David R., L.; Delong, S.; Steve, F.; Gengting, L.; Jim, G.; Stefan, S.; Sebastian, H.; Andreas, D. Approximate Fixed-Point Elementary Function Accelerator for the SpiNNaker-2 Neuromorphic Chip. In Proceedings of the 2018 IEEE 25th Symposium on Computer Arithmetic (ARITH), Amherst, MA, USA, 25–27 June 2018; pp. 37–44. [CrossRef]
10. Yun, Y.S.; Kim, S.; Park, J.; Kim, H.; Jung, J.; Eun, S. Development of Neuromorphic Architecture Integrated Development Environment. In Proceedings of the 2020 International Conference on Green and Human Information Technology (ICGHIT), Hanoi, Vietnam, 5–7 February 2020; pp. 47–49. [CrossRef]
11. Alexandre, S. Neuromorphic microelectronics from devices to hardware systems and applications. *IEICE NOLTA* **2016**, *7*, 468–498. [CrossRef]
12. Xiang, W.; Yongli, H.; Sha, N.; Yi, S.; Qing, W. Biological Band-Pass Filtering Emulated by Oxide-Based Neuromorphic Transistors. *IEEE EDL* **2018**, *39*, 1764–1767. [CrossRef]
13. Zheqi, Y.; Amir, M.A.; Adnan, Z.; Hadi, H.; Muhammad, A.I.; Qammer, H.A. An Overview of Neuromorphic Computing for Artificial Intelligence Enabled Hardware-Based Hopfield Neural Network. *IEEE Access* **2020**, *8*, 2169–3536. [CrossRef]
14. Antara, G.; Rajeev, M.; Virendra, S. Towards Energy Efficient non-von Neumann Architectures for Deep Learning. In Proceedings of the 20th International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 6–7 March 2019; pp. 335–342. [CrossRef]

15. Neeru, S.; Supriya, P.P. Enhancing the Proficiency of Artificial Neural Network on Prediction with GPU. In Proceedings of the 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), Faridabad, India, 14–16 February 2019; pp. 67–71. [CrossRef]
16. Ian, S.; Amirhossein, A. Parallel GPU-Accelerated Spike Sorting. In Proceedings of the 2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE), Edmonton, AB, Canada, 5–8 May 2019; pp. 1–7. [CrossRef]
17. Khomenko, V.; Shyshkov, O.; Radyvonenko, O.; Bokhan, K. Accelerating recurrent neural network training using sequence bucketing and multi-GPU data parallelization. In Proceedings of the 2016 IEEE First International Conference on Data Stream Mining & Processing (DSMP), Lviv, Ukraine, 23–27 August 2016; pp. 100–103. [CrossRef]
18. Mike, D.; Narayan, S.; Lin, T.H.; Gautham, C.; Cao, Y.; Sri, H.C.; Georgios, D.; Prasad, J.; Nabil, I.; Shweta, J.; et al. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* **2018**, *38*, 82–99. [CrossRef]
19. Andrew, Y. Deep Learning Training At Scale Spring Crest Deep Learning Accelerator (Intel® Nervana™ NNP-T). In Proceedings of the 2019 IEEE Hot Chips 31 Symposium (HCS), Cupertino, CA, USA, 15–20 August 2019; pp. 1–20. [CrossRef]
20. Paul, A.M.; John, V.A.; Rodrigo, A.; Andrew, S.C.; Jun, S.; Filipp, A.; Bryan, L.J.; Nabil, I.; Chen, G.; Yutaka, N.; et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* **2014**, *345*, 668–673. [CrossRef]
21. Yu, E.; Cho, S.; Park, B. A Silicon-Compatible Synaptic Transistor Capable of Multiple Synaptic Weights toward Energy-Efficient Neuromorphic Systems. *Electronics* **2019**, *8*, 1102. [CrossRef]
22. Lee, S.M.; Jang, J.H.; Oh, J.H.; Kim, J.K.; Lee, S.E. Design of Hardware Accelerator for Lempel-Ziv 4 (LZ4) Compression. *IEICE Electron. Express* **2017**, *14*, 1–6. [CrossRef]
23. Kim, J.K.; Oh, J.H.; Hwang, G.B.; Gwon, O.S.; Lee, S.E. Design of Low-Power SoC for Wearable Healthcare Device. *JCSC* **2020**, *29*, 1–14. [CrossRef]
24. Bert, M.; Marian, V. An Energy-Efficient Precision-Scalable ConvNet Processor in 40-nm CMOS. *IEEE JSSC* **2017**, *52*, 903–914. [CrossRef]
25. Jang, S.Y.; Oh, J.H.; Yoon, Y.H.; Lee, S.M.; Lee, S.E. Design of a DMA Controller for Augmented Reality in Embedded System. *JKIICE* **2019**, *23*, 822–828. [CrossRef]
26. Lee, J.; Kim, C.; Kang, S.; Shin, D.; Kim, S.; Yoo, H.J. UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision. In Proceedings of the 2018 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 11–15 February 2018; pp. 218–220. [CrossRef]
27. Kang, H.J. Short floating-point representation for convolutional neural network inference. *IEICE Electron. Express* **2019**, *16*, 1–11. [CrossRef]
28. Kota, A.; Kodai, U.; Yuka, O.; Kazutoshi, H.; Ryota, U.; Takumi, K.; Masayuki, I.; Tetsuya, A.; Shinya, T.Y.; Masato, M. Dither NN: Hardware/Algorithm Co-Design for Accurate Quantized Neural Networks. *IEICE Trans. Inf. Syst.* **2019**, *E102.D*, 2341–2353. [CrossRef]
29. Kim, J.K.; Oh, J.H.; Yang, J.H.; Lee, S.E. 2D Line Draw Hardware Accelerator for Tiny Embedded Processor in Consumer Electronics. In Proceedings of the 2019 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 11–13 January 2019; pp. 1–2. [CrossRef]
30. Mengyue, Z.; Weihan, L.; Jianlian, Z.; Huisheng, G.; Fanyi, W.; Bin, L. Embedded Face Recognition System Based on Multi-task Convolutional Neural Network and LBP Features. In Proceedings of the 2019 IEEE International Conference of Intelligent Applied Systems on Engineering (ICIASE), Fuzhou, China, 26–29 April 2019; pp. 132–135. [CrossRef]
31. Kim, Y.H.; An, G.J.; Sunwoo, M.H. CASA: A Convolution Accelerator using Skip Algorithm for Deep Neural Network. In Proceedings of the 2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 26–29 May 2019; pp. 1–5. [CrossRef]
32. Bo, Y.; John, P.; Lu, L.; Nick, A.; Yao, L. An Inductive Content-Augmented Network Embedding Model for Edge Artificial Intelligence. *IEEE Trans. Ind. Inf.* **2019**, *15*, 4249–4305. [CrossRef]
33. Fengbin, T.; Shouyi, Y.; Peng, O.; Shibin, T.; Leibo, L.; Shaojun, W. Deep Convolutional Neural Network Architecture With Reconfigurable Computation Patterns. *IEEE Trans. VLSI* **2017**, *25*, 2220–2233. [CrossRef]
34. Pete, W. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv* **2018**, arXiv:1804.03209.

35. Yann, L.; L´eon, B.; Yoshua, B.; Patrick, H. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
36. Schafer, R.; Rabiner, L. Design and simulation of a speech analysis-synthesis system based on short-time Fourier analysis. *IEEE Trans. Audio Electroacoust.* **1973**, *21*, 165–174. [CrossRef]
37. Viola, P.; Jones, M. Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Kauai, HI, USA, 8–14 December 2001; pp. 511–518. [CrossRef]
38. Canny, J. A Computational Approach to Edge Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **1986**, *PAMI-8*, 679–698. [CrossRef]
39. Steve, B.F.; Francesco, G.; Steve, T.; Luis, A.P. The SpiNNaker Project. *Proc. IEEE* **2014**, *102*, 652–665. [CrossRef]
40. Arnon, A.; Brian, T.; David, B.; Timothy, M.; Jeffrey, M.; Carmelo, D.N.; Tapan, N.; Alexander, A.; Guillaume, G.; Marcela, M.; et al. A Low Power, Fully Event-Based Gesture Recognition System. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 7388–7397. [CrossRef]
41. Yoon, Y.H.; Jang, S.Y.; Choi, D.Y.; Lee, S.E. Flexible Embedded AI System with High-speed Neuromorphic Controller. In Proceedings of the 2019 International SoC Design Conference (ISOCC), Jeju, Korea, 6–9 October 2019; pp. 265–266. [CrossRef]