IEICE TRANS. INF. & SYST., VOL.E104–D, NO.8 AUGUST 2021

PAPER Special Section on Multiple-Valued Logic and VLSI Computing

Classification Functions for Handwritten Digit Recognition

Tsutomu SASAO[†a)], ***Member***, **Yuto HORIKAWA**[†], ***Nonmember***, ***and*** **Yukihiro IGUCHI**[†], ***Member***

SUMMARY A classification function maps a set of vectors into several classes. A machine learning problem is treated as a design problem for partially defined classification functions. To realize classification functions for MNIST hand written digits, three different architectures are considered: Single-unit realization, 45-unit realization, and 45-unit $\times r$ realization. The 45-unit realization consists of 45 ternary classifiers, 10 counters, and a max selector. Test accuracy of these architectures are compared using MNIST data set.

key words: *linear decomposition, partially defined function, support minimization, classification, digit recognition, MNIST, index generation function, machine learning, neural network, ensemble method*

1. Introduction

Given disjoint sets of elements, the problem to find a simple rule to distinguish these sets is a major topic of machine learning and data mining. A **partially defined classification function** is the mapping:

$$f : D \to \{1, 2, \ldots, m\},$$

where $D \subset \{0,1\}^n$ represents the **training set**. When the number of elements in the training set $|D|$ is much smaller than the total number of input combinations 2^n, the original function f can be represented by compound variables y_j as follows:

$$f(x_1, x_2, \ldots, x_n) = g(y_1, y_2 \ldots y_p), \tag{1}$$

where g is a **reduced classification function** of p variables, and y_j $(j = 1, 2, \ldots, p)$ are linear functions of the input variable $x_1, x_2, \ldots, x_n$:

$$y_j = a_1 x_1 \oplus a_2 x_2 \oplus \cdots \oplus a_n x_n,$$

where $a_i \in \{0, 1\}$, and $p < n$.

Interestingly, the reduced classification function g produces correct responses not only for the training set, but also for much of the unknown **test set**. For the real data, such as handwritten digits, the reduced classification function correctly recognizes much of the test set. That is, the reduced classification function g has a **generalization ability** [5]. Although the test accuracy based on reduced classification functions is lower than that of neural networks, the presented method requires no complex learning. So, it is promising for simple image recognition.

The rest of this paper is organized as follows: Sect. 2 introduces classification functions, describes compound variables and their reduction method; Sect. 3 shows benchmark functions; Sect. 4 shows the single-unit realization; Sect. 5 shows the 45-unit realization; Sect. 6 shows the 45-unit $\times r$ realization; Sect. 7 compares different architectures; Sect. 8 shows methods to implement counters and the max selector; and Sect. 10 concludes the paper.

A preliminary version of this paper was presented as [16]. In [16], threshold elements are used in the output parts, while in this paper, counters and max selectors are used. With this modification, the accuracy has been improved significantly.

2. Classification Functions and Their Realization

Definition 2.1: Consider the set of k distinct vectors of n bits. These vectors are **registered vectors**. In the framework of machine learning, the set of registered vectors corresponds to the **training set**. To each registered vector, assign an integer between 1 and m, where $2 \le m \le k$. The **registered vector table** shows the corresponding function values for the registered vectors. A **partially defined classification function** produces the corresponding function values for the input vectors that match the registered vectors. When the input vector does not match the registered vectors, the function value is undefined. A partially defined classification function represents a mapping $f : D \to \{1, 2, \ldots, m\}$, where $D \subset B^n$ is the set of registered vectors, and $B = \{0, 1\}$. k is the **weight** of the function.

Example 2.1: Table 1 is a registered vector table of the classification function with weight $k = 10$ and $m = 2$. ■

Partially defined functions often can be represented

Table 1 Registered vector table

x_1	x_2	x_3	x_4	x_5	x_6	x_7	f
1	1	1	0	1	1	1	1
1	0	1	0	1	0	0	1
1	0	0	0	0	1	0	1
0	1	1	0	0	1	1	1
0	0	1	1	0	1	1	1
0	1	1	1	0	1	1	2
0	1	0	1	1	1	1	2
0	1	0	0	0	1	1	2
0	0	0	1	0	1	0	2
0	0	0	0	1	0	1	2

Manuscript received August 22, 2020.
Manuscript revised February 11, 2021.
Manuscript publicized April 1, 2021.

[†]The author is with the Department of Computer Science, Meiji University, Kawasaki-shi, 214–8571 Japan.
a) E-mail: sasao@ieee.org

DOI: 10.1587/transinf.2020LOP0002

with fewer variables by using **linear decompositions** [13]. In the linear decomposition shown in Fig. 1, L denotes a linear function, while G denotes a general function (in most cases, non-linear function). We assume that the cost of the linear part is $O(np)$, while the cost of the general part is $O(q2^p)$.

Definition 2.2: **Compound variables** have the form $y = c_1x_1 \oplus c_2x_2 \oplus \cdots \oplus c_nx_n$, where $c_i \in \{0, 1\}$. The **compound degree** of the variable y is $\sum_{i=1}^{n} c_i$, where $\sum$ denotes an integer addition. **Primitive variables** are variables with compound degree 1.

Definition 2.3: Given a partially defined function f, the linear transformation that minimizes the number of the compound variables is an **optimal transformation**.

When the number of compound variables can be reduced to $q = \lceil \log_2 m \rceil$ by a linear transformation, then the transformation is optimum.

Example 2.2: The function shown in Table 1, can be represented as follows:

When primitive variables are used, the function can be represented with four variables:

$$\begin{aligned} f &= (x_1x_2x_3\bar{x}_4 \vee x_1\bar{x}_2x_3\bar{x}_4 \vee x_1\bar{x}_2\bar{x}_3\bar{x}_4 \vee \bar{x}_1x_2x_3\bar{x}_4 \vee \\ &\quad \bar{x}_1\bar{x}_2x_3x_4) \vee 2(\bar{x}_1x_2x_3x_4 \vee \bar{x}_1x_2\bar{x}_3x_4 \vee \\ &\quad \bar{x}_1x_2\bar{x}_3\bar{x}_4 \vee \bar{x}_1\bar{x}_2\bar{x}_3x_4 \vee \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4) \\ &= (x_2x_3\bar{x}_4 \vee x_1\bar{x}_2\bar{x}_4 \vee \bar{x}_1\bar{x}_2x_3x_4) \vee 2\bar{x}_1(\bar{x}_3 \vee x_2x_4), \end{aligned}$$

where $\vee$ denotes the max operation, and the concatenation denotes the min operation.

When the compound variables $y_1 = x_1$ and $y_2 = x_3$, and $y_3 = x_2 \oplus x_4$ are used, the function can be represented with only three variables:

$$f = (y_1\bar{y}_3 \vee y_2y_3) \vee 2\bar{y}_1(\bar{y}_2 \vee \bar{y}_3).$$

■

The reduction methods for primitive variables are well known [7], [11]. However, nobody has ever minimized problems with $n = 784$ variables successfully. We developed a special algorithm for this purpose using the notion of impurity measure [14]. The reduction problem for compound variables is to find a linear decomposition that minimizes the intermediate variables p shown in Fig. 1. Recently, we developed an efficient algorithm to find a good linear decomposition. With this, we can design a compact circuit within a reasonable computation time. This algorithm is shown in [15].

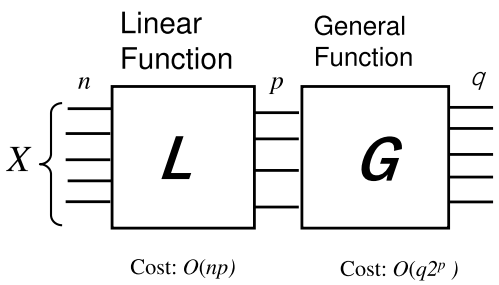

Fig. 1 Linear decomposition

3. Benchmark Function

Benchmark function were generated from the MNIST [18] data set of handwritten digits. The data in MNIST consists of bit maps of 28×28 images, the training set consist of 6×10^4 images, while the test set consists of 10^4 images. They are grayscale images, but we converted them into binary ones, by setting the threshold 96. In this way, we had an n-variable m-valued classification function, where $n = 28 \times 28 = 784$, and $m = 10$. Also in this process, we removed duplicated data. Table 2 shows the size of the training set and the test set, after removing duplicated data.

To evaluate the performance of a classifier, we use:

Definition 3.4:

$$Accuracy = \frac{\text{Number of correctly recognized images}}{\text{Total number of images.}}$$

The **training accuracy** is calculated by using images in the training set, while the **test accuracy** is calculated by using images in the test set.

4. Single-Unit Realization

A **single-unit realization** is implemented by a cascade of a linear circuit and a memory, as shown in Fig. 2. When primitive variables are used, the linear part can be omitted. In such a case, the function can be implemented by a single memory.

The number of primitive variables was reduced to $p = 37$ by a heuristic algorithm in [14]. Then, the number of compound variables was reduced to $p = 25$ by a algorithm for linear decomposition in [15].

The reduced classification function g correctly recognized all the images in the training set. Next, we applied the images in the test set shown in Table 2 to the reduced classification function g, and checked if g recognized the images

Table 2 Sizes of training and test sets.

Data	# of samples
Training Set	59981
Test Set	9993

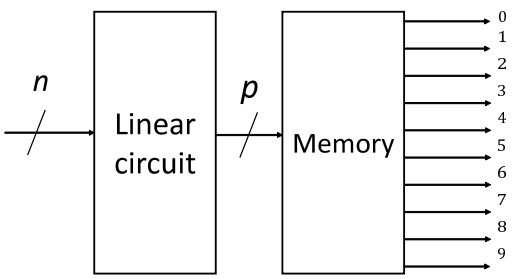

Fig. 2 Single-unit realization

Table 3 Recognition result for single-unit realization.

Result	Primitive variables	Compound variables
	$p = 37$	$p = 25$
Correctly recognized	1462	1561
Incorrectly recognized	9	17
Unrecognized	8522	8415
Total	9993	9993
Test Accuracy	0.146	0.156

in the test set correctly or not. Table 3 shows the results. When the images were unrecognized, the circuit for g produced **unrecognized** output, represented by the $(0, 0, \ldots, 0)$ vector. Table 3 shows that the test accuracy of the reduced classification function g is 0.146. If we do not reduce the variables, but use a memory with 784 inputs, then the test accuracy would be 0.0. However, the memory with reduced variables correctly predicted the values for considerable part of unknown test data. For example, when variables are reduced to 37 primitive variables, the memory with reduced variables recognizes $2^{784-37} = 2^{747} \simeq 1.826 \times 10^{224}$ times more images than the images in the training data set.

In the case of MNIST data set, the probability of "if the values of 37 variables are the same for a test image and a training image, then two images represent the same digit," is higher than 0.1, the probability obtained by a random guess.

Although this test accuracy is much lower than that of the neural networks [18], it is an important property.

Theorem 4.1: The single-unit realization always produces correct results for the training data.

(Proof) The circuit is designed so that it correctly recognizes all the images in the training set. □

Thus, the accuracy for the training data is 1.00.

5. 45-Unit Realization

Although the single-unit realization is simple, its test accuracy is very low. In this section, to improve the test accuracy, we introduce the **45-unit realization** shown in Fig. 3. In this method, for each pair of digits, we use a **unit**. In the circuit, each unit decides if the input image represents the digit i, or the digit j, or another digit. With $\binom{10}{2} = 45$ such units, we can make a final decision using a majority vote.

In Fig. 3, a square symbol denotes a unit. The unit i/j has two outputs: The output $(1, 0)$ denotes that the input image represents the digit i; the output $(0, 1)$ denotes that the input image represents the digit j; and the output $(0, 0)$ denotes that the input image represents another digit or unknown. Thus, each unit produces a **ternary output**. Since there are 45 units, the total number of outputs is 90. Figure 3 shows the circuits for only digits 0, 1 and 2. Circuits for other digits are omitted. In addition, we use 10 **counters**, shown by $\sum$ symbols in Fig. 3. The i-th counter has 9 inputs with label i, and counts the number of 1's in the inputs, and represents it by a 4-bit binary number.

Example 5.3: Assume that a training image representing

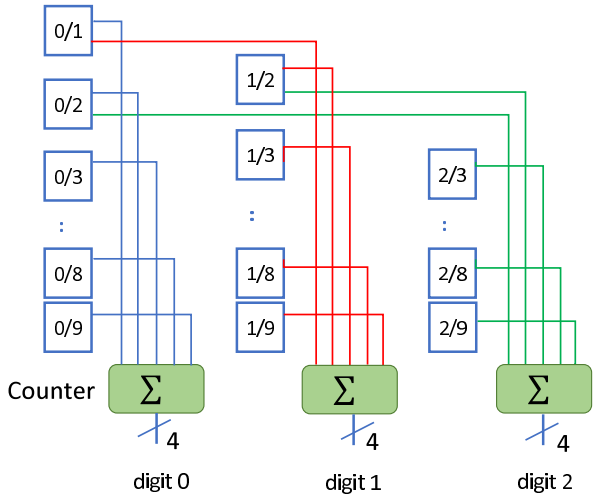

Fig. 3 45-unit realization.

Table 4 Recognition result for 45-unit realization.

Result	Primitive variables	Compound variables
Correctly recognized	8773.5	8695.6
Incorrectly recognized	1219.5	1297.4
Total	9993.0	9993.0
Test Accuracy	0.878	0.870

'0' is applied to the circuit in Fig. 3. Then, all the units in the leftmost column recognize the digit 0, and all the blue lines represent 1. Thus, all the inputs to the counter for the digit 0 become 1. So, it receives 9 votes. In this case, in the top unit in the leftmost column labelled 0/1, the red output line becomes 0. So, the first input of the counter for the digit 1 becomes 0. So, it received votes less than 9. Similarly, for other counters, at least one input is 0, and the number of votes is less than 9. ■

We also use a **max selector** that selects the digit with the largest count, which is not shown in Fig. 3. Details of the counters and the max selector are shown in Sect. 8.

Table 4 shows recognition results of the 45-unit realization. To design the unit i/j, only the training images of the digits i and j are used. This drastically reduced the size of memory necessary to implement each unit.

The 45-unit realization produces much higher test accuracy than the single-unit realization. The next theorem shows that the training accuracy for the 45-unit realization is 1.00.

Theorem 5.2: The 45-unit realization always produces correct results for the images in the training set.

(Proof) The number of inputs to each counter is 9. For a training image representing the digit '0', the value of the counter for the digit '0' is 9. On the other hand, the values of the other counters are less than 9. Thus, any training image for the digit 0 produces correct result. This is true for other digits. □

When an unknown test data is applied, the majority

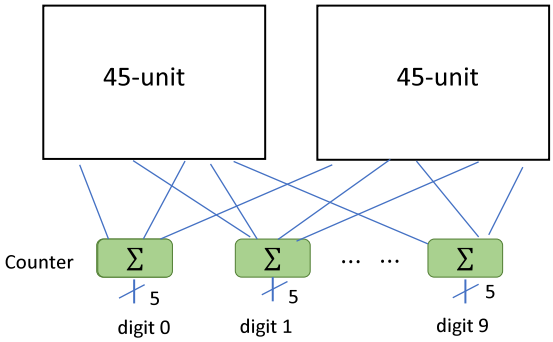

Fig. 4 45-unit ×2 realization.

Table 5 Recognition result for 45-unit ×2 realization.

Result	Primitive variables	Compound variables
Correctly recognized	8955.0	8901.1
Incorrectly recognized	1038.0	1091.9
Total	9993.0	9993.0
Test Accuracy	0.896	0.891

Table 6 Recognition result for 45-unit ×4 realization.

Result	Primitive variables	Compound variables
Correctly recognized	9062.8	9044.4
Incorrectly recognized	930.2	948.6
Total	9993.0	9993.0
Test Accuracy	0.907	0.905

vote may fail. When there are s counters with the maximum values, the correct answers is counted as $1/s$, while the incorrect answers is counted as $1-(1/s)$. Since there always exists a counter with the maximum value, images are always recognized either correctly or incorrectly.

6. 45-Unit ×r Realization

In the previous section, the 45-unit realization was used to improve the test accuracy. To further improve the test accuracy, the training data is partitioned into r groups of similar sizes, and for each group, digits are recognized by a 45-unit, independently. And, finally, the max selector is used to find the largest vote. Figure 4 illustrates the **45-unit ×r realization**, where $r = 2$. In this case, each module produces $45 \times 2 = 90$ outputs. Thus, the total number of outputs is $45 \times 2 \times 2 = 180$. The i-th counter has $9 \times 2 = 18$ inputs with label i.

This is a **simple ensemble method**: r weak classifiers are combined to make a stronger classifier. The 45-unit ×r realizations improve the test accuracy, as well as reduce the total amount of memory. Unfortunately, the training accuracy can be decreased. For example, when $r = 2$, the training accuracy was 0.997 when primitive variables were used, and was 0.995 when compound variables were used. This occurs when at least one unit in Fig. 4 incorrectly reconize the image.

Tables 5 to 8 show the numbers of correctly recognized images for 45-unit ×r realizations. Note that the test accuracy takes its maximum, when $r = 4$.

Table 7 Recognition result for 45-unit ×8 realization.

Result	Primitive variables	Compound variables
Correctly recognized	9024.8	9027.2
Incorrectly recognized	968.2	965.8
Total	9993.0	9993.0
Test Accuracy	0.903	0.903

Table 8 Recognition result for 45-unit ×16 realization.

Result	Primitive variables	Compound variables
Correctly recognized	8997.5	8981.3
Incorrectly recognized	995.5	1011.7
Total	9993.0	9993.0
Test Accuracy	0.900	0.899

Table 9 Test Accuracy for different values of r.

Architecture	Primitive variables	Compound variables
45-unit×1	0.878	0.870
45-unit×2	0.896	0.891
45-unit×4	0.907	0.905
45-unit×8	0.903	0.903
45-unit×16	0.900	0.899

7. Comparisons of Architectures

In this section, we compare 45-unit ×r architectures for different values of r.

7.1 Test Accuracy

Table 9 compares test accuracy for different r.

The test accuracy of 45-unit ×r realizations increases with the value of r, until $r = 4$. Note that the 45-unit ×4 realization produced the maximal test accuracy.

This can be interpreted as follows: The total number of training vectors is 59981. Since there are 10 digits, each digits has 5998.1 vectors, on the average. So, each of the 45-unit ×r realizations is trained by $5998.1 \times 2/r$ vectors, on the average. When $r = 4$, this value is about 3000. This may be the minimum number of training vectors to produce good test accuracy for the given benchmark function.

7.2 Number of Variables and Memory Requirement

Table 10 compares the average number of input variables for each unit.

Let k be the number of sample images in the training set. Then, the number of variables to represent the classification function is at most $\lceil 2 \log_2 k \rceil - 2$ [13]. When the input data is partitioned into two groups of similar sizes, we can expect that the number of input variables is reduced. Experimental results shown in Table 10 confirm that there is a reduction by 2.0.

Table 10 Average number of variables for each unit.

	Primitive variables	Compound variables
45-unit×1	19.42	16.86
45-unit×2	17.04	15.03
45-unit×4	15.08	12.77
45-unit×8	12.34	11.43
45-unit×16	10.18	9.60

Table 11 Memory sizes for different architecturs (Kilo bits).

Architecture	Primitive variables	Compound variables
45-unit	197,935	15,901
45-unit×2	68,108	9,167
45-unit×4	18,830	5,321
45-unit×8	7,093	3,068
45-unit×16	2,814	1,665

Also, CPU time for compound variable reduction for larger r is much shorter, since the CPU time for variable minimization is proportional to $k_{i,j}^2$, where $k_{i,j}$ is the number of vectors to train the unit i/j, and $k_{i,j}$ is, on the average, $\frac{k}{5r}$.

The 45-unit realization requires LUTs with

$$\sum_{i=0}^{8}\sum_{j=i+1}^{9}\lceil\log_2(2+1)\rceil 2^{p_{ij}}$$

bits, where p_{ij} denotes the number of the variables for the unit i/j.

Table 11 compares the memory sizes for different r. It shows that the memory sizes[†] of 45-unit $\times r$ realizations decrease with the value of r.

In addition, the 45-unit $\times r$ realization requires 10 **counters** with $90 \times r$ inputs, as well as the **max selector** that selects the digit with the largest count. The costs for counters and the max selector are not included in Table 11.

8. Counter and Max selector

In this part, we consider implementations of counters and a max selector.

8.1 Counter

A **counter**, also called as a **pop-counter** (population counter) [21], or a **compressor** [10], counts the number of 1's in the inputs, and represents it by a natural binary number. It can be implemented as a tree of full adders. Since the counter is a basic element in arithmetic circuits, extensive research has been done. For example, [8] considers ASIC realizations, while [10] and [21] consider FPGA realizations. When the performance is not critical, the counter can be implemented by a sequential circuit.

[†]For 45-unit ×4 realization, when compound variables are used, the average number of variable is 12.77. This means that most units in Fig. 3 can be realized by block RAMs. For example, Xilinx FPGA [22] contains block RAMs, each can be configured as a pair of 18Kb RAMs or a 36Kb RAM. So, each can be used as either a pair of 14-input LUTs or a 15-input LUT. ZU6CG contains 714 BRAMs, and ZU9CG contains 912 BRAMs. Note that the 45-unit ×4 realization requires 180 units.

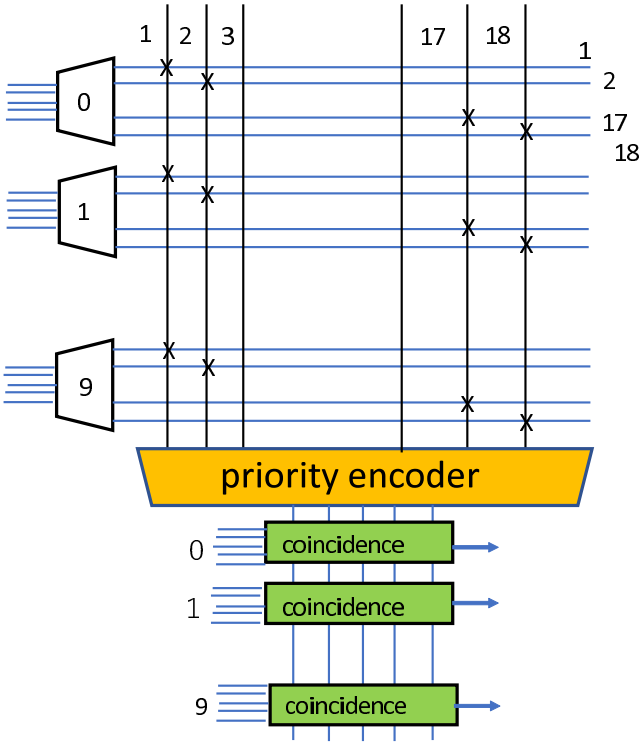

Fig. 5 Max selector

8.2 Max Selector

The work for max selectors is rare, so we show the detail of the design.

Consider the 45-unit × 2 realization shown in Fig. 4. It has 45×2 units, and $45 \times 2 \times 2 = 180$ outputs. Also, it uses 10 counters with 18 inputs and 5 outputs.

Figure 5 shows the **max selector** for the 45-unit × 2 realization. It consists of 10 decoders, one priority encoder, and 10 coincidence circuits.

Each **decoder** has 5 inputs and 18 outputs. The input of a decoder denotes the number of 1's produced by a counter. The output of a decoder denotes the number of 1's represented by the 1-out-of-18 code.

The matrix has 180 rows and 18 columns. A × mark denotes an OR connection. So, if one of the row connected with × is 1 (high), the corresponding column will be 1 (high). Thus, the matrix works as an **OR array**. In the OR array, 18 OR gates with 10 inputs are realized.

The **priority encoder** detects the right-most column that is high, and produces the binary representation of the largest number. Thus, the priority encoder finds the largest value produced by the counters.

Finally, the **coincidence circuits** find the digit that produced the largest count.

When the performance is not critical, the max selector

can be implemented by a sequential circuit.

9. Comparison with Neural Networks

9.1 FPGA Implmentation

Most neural networks assume signals with real numbers. So, processor-based implementations are common. Thus, they are several orders of magnitude slower than LUT based-one.

For faster applications, binarized neural networks are used. They require conversion from original neural networks into binary. Their test accuracies can be comparable to original neural networks. For example, FINN-R MLP-4 on AWS F1 [2], is a fully binarized multilayered perceptron. It uses 1,652 BRAM18s and 337,753 LUTs, and achieves test accuracy 0.977. On the other hand, the 45-unit ×4 realization uses $45 \times 4 = 180$ units. When compound variables are used, 4 units require 16 inputs, 35 units require 15 inputs, and the remaining 141 units require at most 14 inputs. Since a unit with 15 (16) inputs can be synthesized with two (four) 14-input LUTs, the network can be mapped into $227 \times 2 = 454$ BRAM18s. Note that each unit has two outputs. This shows that our realization require fewer BRAMs than FINN-R.

Since, each unit requires only one BRAM access, it is faster than FINN-R. Also, it requires lower power, since our network requires fewer BRAMs than FINN-R. Note that this comparison excludes the softmax part [20].

9.2 Advantage vs. Disadvantage

Advantages The design flow of the proposed method is straightforward. It directly produces the circuit that recognizes the training set. No need to train neural networks. No need to convert them to binary ones. The proposed method assume LUTs as basic elements, and try to reduce the number of inputs for LUTs. Thus, the method is suitable for memory-based implementations, including FPGA.

Disadvantage The accuracy is not so good as neural networks. This is due to the simplicity of the topology. Also, recognition of complex images seems to be difficult, since no convolution layers are used.

10. Concluding Remarks

In this paper, we introduced classification functions for machine learning. The reduced classification function correctly recognizes not only most of the training images, but also much of the test images. Our method is to consider the training set as a partially defined function, and to design the simplest circuit that satisfies the given specification. Our measure of the simplicity is the number of variables. Thus, the method is useful for LUT-based implementations.

An ensemble method can be used to improve the test accuracy. That is, to partition the input data into groups, and to derive the classifier for each group, and finally, to combine the results by the max selector. The ensemble method reduces the amount of memory necessary to implement the circuit.

The contributions of this paper are:

1. A new design method for classifiers: Find a classification function with the fewest variables that satisfies the training data, and implement it by LUTs.
2. A method to improve test accuracy: Partition the training data into groups, and implement each by a LUT, and combine them by counters and the max selector.

Using these methods, we can design a compact network with a required accuracy.

Our method is suitable for simple image recognition such as binary character or symbol recognition. The merit is its simplicity. We have similar results for fashion-MNIST [6].

Acknowledgments

This research is partly supported by the grant of the Japan Society for the Promotion of Science (JSPS), Grant in Aid for Scientific Research. Dr. Alan Mishchenko of University California, Berkeley, and Dr. Satrajit Chatterjee of Google AI gave us useful comments. Prof. Jon T. Butler improved presentation of the paper.

References

[1] J.T. Astola, P. Astola, R.S. Stankovic, and I. Tabus, "Algebraic and combinatorial methods for reducing the number of variables of partially defined discrete functions," International Symposium on Multiple-Valued Logic (ISMVL 2017), Sapporo, pp.167–172, May 2017.
[2] M. Blott, T.B. Preuser, N.J. Fraser, G. Gambardella, K.O. Brien, Y. Umuroglu, M. Leeser, and K. Vissers, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," ACM Trans. Reconfigurable Technol. Syst., vol.11, no.3, pp.1–23, Dec. 2018.
[3] A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth, "Occam's Razor," Inform. Process. Lett., vol.24, no.6, pp.377–380, 1987.
[4] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, Classification and Regression Trees, CRC Press, New York, 1984.
[5] S. Chatterjee, "Learning and memorization," International Conference on Machine Learning (ICML 2018), Stockholm, Sweden, pp.754–762, July 10-15, 2018.
[6] https://www.kaggle.com/zalando-research/fashionmnist
[7] T. Ibaraki, "Partially defined Boolean functions," Chapter 8 in: Y. Crama and P.L. Hammer, Boolean Functions - Theory, Algorithms and Applications, Cambridge University Press, New York, 2011.
[8] R.F. Jones and E.E. Swartzlander, "Parallel counter implementation," Journal of VLSI Signal Processing, vol.7, no.3, pp.223–232, 1994.
[9] A. Krizhevsky, "Learning multiple layers of features from tiny images," Master's thesis, University of Toronto, 2009.
[10] M. Kumm and J. Kappauf, "Advanced compressor tree synthesis for FPGAs," IEEE Trans. Comput., vol.67, no.8, pp.1078–1091, Aug. 2018.
[11] J. Kuntzmann, Algèbre de Boole, Dunod, Paris, 1965. English translation: Fundamental Boolean Algebra, Blackie and Son Limited, London and Glasgow, 1967.
[12] A.L. Oliveira and A. Sangiovanni-Vincentelli, "Learning complex

boolean functions:Algorithms and applications," Advances in Neural Information Processing Systems, no.6, pp.911–918. Morgan-Kaufmann, 1994.
[13] T. Sasao, Index Generation Functions, Morgan & Claypool, Oct. 2019.
[14] T. Sasao, "Reduction methods of variables for large-scale classification functions," in International Workshop on Logic and Synthesis (IWLS-2020), pp.82–87, July 27-30, 2020.
[15] T. Sasao, "On the minimization of variables to represent partially defined classification functions," International Symposium on Multiple-Valued Logic, (ISMVL-2020), pp.117–123, Nov. 9-11, Japan.
[16] T. Sasao, Y. Horikawa, and Y. Iguchi, "Handwritten digit recognition based on classification functions," International Symposium on Multiple-Valued Logic, (ISMVL-2020), pp.124–129, Nov. 9-11, Japan.
[17] D.A. Simovici, M. Zimand, and D. Pletea, "Several remarks on index generation functions," International Symposium on Multiple-Valued Logic (ISMVL-2012), Victoria, Canada, pp.179–184, May 2012.
[18] http://yann.lecun.com/exdb/mnist/
[19] A. Tapp, "A new approach in machine learning," (Preliminary report), Sept. 16, 2014. https://arxiv.org/abs/1409.4044
[20] Y. Umuroglu, Y. Akhauri, N.J. Fraser, and M. Blott. "LogicNets: Co-designed neural networks and circuits for extreme-throughput applications," 30th International Conference on Field-Programmable Logic and Applications, pp.291–297, May 2020.
[21] E. Wang, J.J. Davis, P.Y.K. Cheung, and G.A. Constantinides, "LUTNet: Rethinking inference in FPGA soft logic," FCCM 2019, pp.26–34.
[22] https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

Tsutomu Sasao received the B.E., M.E., and Ph.D. degrees in Electronics Engineering from Osaka University, Osaka Japan, in 1972, 1974, and 1977, respectively. He has held faculty/research positions at Osaka University, Japan; IBM T. J. Watson Research Center, Yorktown Height, NY; the Naval Postgraduate School, Monterey, CA; and Kyushu Institute of Technology, Iizuka, Japan; and Meiji University, Kawasaki, Japan. Now, he is a visiting researcher at Meiji University, Kawasaki, Japan. His research areas include logic design and switching theory, representations of logic functions, and multiple-valued logic. He has published more than 9 books on logic design including, Logic Synthesis and Optimization, Representation of Discrete Functions, Switching Theory for Logic Synthesis, Logic Synthesis and Verification, Memory-Based Logic Synthesis, and Index Generation Functions, in 1993, 1996, 1999, 2001, 2011, and 2019, respectively. He has served Program Chairman for the IEEE International Symposium on Multiple-Valued Logic (ISMVL) many times. Also, he was the Symposium Chairman of the 28th ISMVL held in Fukuoka, Japan in 1998. He received the NIWA Memorial Award in 1979, Takeda Techno-Entrepreneurship Award in 2001, and Distinctive Contribution Awards from IEEE Computer Society MVL-TC for papers presented at ISMVLs in 1986, 1996, 2003, 2004, 2012 and 2019. He has served an associate editor of the IEEE Transactions on Computers. He is a Life Fellow of the IEEE.

Yuto Horikawa received the B.E. degree in computer science from Meiji University in 2020. He is now a Mater Student of the same university.

Yukihiro Iguchi received the B.E, M.E., and Ph.D. degrees in electronic engineering from Meiji University, Kanagawa, Japan, in 1982, 1984, and 1987, respectively. He is now a professor of Meiji University. His research interest includes logic design, switching theory, reconfigurable systems, and electric vehicles. In 1996 and 2006, he spent each year at Kyushu Institute of Technology. He received Takeda Techno-Entrepreneurship Award in 2001.