2015 RISP International Workshop on Nonlinear Circuits,
Communications and Signal Processing (NCSP'15)
Kuala Lumpur, Malaysia, February 27 - March 2, 2015

A New Architecture for Feature Extraction to Perform Machine Learning by using Motion Vectors and its Implementation in an FPGA

Toshiyuki Itou[1], Masafumi Mori[1], Masayuki Ikebe[1], Tetsuya Asai[1],
Tadahiro Kuroda[2], and Masato Motomura[1]

[1]Hokkaido University
Kita 14, Nishi 9, Kita–ku, Sapporo, Hokkaido, Japan
Phone/FAX:+81 11-706-(7147)
E-mail: itou@lalsie.ist.hokudai.ac.jp

[2]Keio University
3–14–1 Hiyoshi, Kohoku-ku, Yokohama, Kanagawa, Japan

Abstract

In this study, we propose a machine learning architecture by using motion vectors that were estimated exploiting high-speed imaging. The motion vector fields are estimated from images captured by a high-speed camera and performed feature extraction, and then normalization and scaling, before they are utilized as inputs in a neural network called a simple perceptron. A support vector machine is used for learning with the same linear classifier as the simple perceptron and its synaptic weights are then employed by the simple perceptron. The simple perceptron classifies images into two types, i.e., dangerous (approaching objects) or safe (non-approaching objects). We describe the architecture based on machine learning, which uses estimated motion vectors obtained by high-speed imaging. We also present the results obtained after the implementation of the architecture in an FPGA.

1. Introduction

Recently, image processors have been developed that can be installed in portable terminals such as smartphones [1]. The traditional architectures used for complex processing connect multiple processors or memory components on a printed circuit board. However, this method leads to a significant increase in the size of the board and the data rate between devices is very slow [2]. Thus, a method was devised that integrates several chips and connects them in three dimensions [3]. The chip area is reduced using this method and the data rate is improved between the chips. We expect that the data rate will be increased considerably in the near future, thereby developing a new field of modern semiconductor applications. For example, we may assume that images captured by an image sensor at 1000 fps will be transmitted directly to an image processor. In this case, the inter-frame differences will become smaller as the video frame rate increases, thereby decreasing the search ranges used for motion vector estimation by block-matching. Based on this fact, we proposed an architecture that estimated motion vectors with a small number of calculations [4]. We used the motion vectors estimated by the architecture for machine learning.
The motion vector fields were estimated by feature extraction and used as inputs by a neural network in our proposed architecture for machine learning to classify images into two classes i.e., dangerous (approaching objects) or safe (non-approaching objects). We used a neural network called a simple perceptron [5] but it could not solve nonlinear separable problems [6]. Therefore, the input data must be made linearly separable by feature extraction before the data can be used inputs by the simple perceptron. Thus, in our proposed feature extraction method, the estimated motion vector fields are divided into several areas and the sums of the vector sizes are calculated in each direction (nine directions: up, upper right, right, lower right, down, lower left, left, upper left, and motionlessness). Furthermore, the summed vector sizes are calculated for each combination of two vectors (a total of 28 combinations, e.g., a combination of up and upper right). This feature extraction method is based on Poggio's HMAX model [7][8]. The classification of images became possible by the feature extraction method [4]. The values calculated by the feature extraction method are normalized and scaled from –1 to 1, before they are used as inputs by the simple perceptron. A high capacity for generalization is required during classification using machine learning. Thus, a support vector machine [9] is learned in advance with the same linear classifier as the simple perceptron. After learning, the support vector machine's synaptic weights are used by the simple perceptron. We implemented this architecture in an FPGA.

2. Algorithm and architecture for machine learning

2.1 Feature extraction to learn the motion vectors

We use a simple perceptron for machine learning. However, it cannot solve nonlinear separable problems. Thus, the

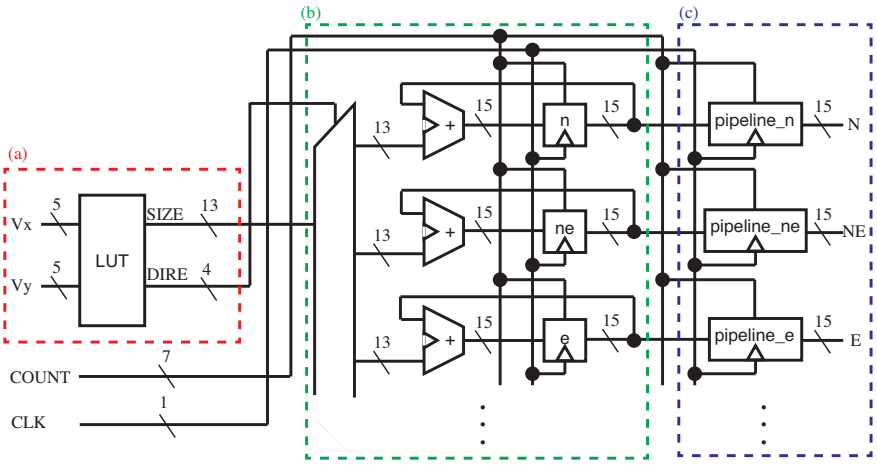

Figure 1: Block diagram of the architecture used for feature extraction

input data must be made linearly separable by feature extraction before they can be used as inputs by the simple perceptron.

In our proposed feature extraction method, the estimated motion vector fields are divided into several areas and the sums of the vector sizes are calculated in each direction (nine directions: up, upper right, right, lower right, down, lower left, left, upper left, and motionlessness). Furthermore, the summed vector sizes are calculated for each combination of two vectors (a total of 28 combinations, e.g., a combination of up and upper right). Initially, the X component and Y component of the first motion vector in a specific area are used as inputs. The size and direction of the motion vector are calculated from the inputs, and the size is added to the current summed size of the direction. Similarly, the size is added to each current summed size for combinations that include the direction. For example, if the size of the input motion vector is 0.4 and its direction is up, 0.4 is added to the current summed size for the up direction. In addition, 0.4 is added to each summed size for the combinations that include up, e.g., the combination of up and upper right. The same calculations are performed for all of the motion vectors in the specific area. Next, the calculated summed vector size for one direction and combinations of two directions are applied to all areas. This feature extraction method is based on Poggio's HMAX model.

Figure 1 shows a block diagram of the feature extraction architecture. First, the X and Y components of the estimated and integrated motion vectors are inserted in a look-up table (LUT), as shown in Figure 1(a). The LUT transforms the X and Y components into a vector size (SIZE in Figure 1(a)) and a direction (DIRE in Figure 1(a)). The SIZE is a 13-bit fixed point number, where the integer part is 2-bit with a sign bit and the fractional part is 11-bit. This is because the maximum size of the motion vectors is 1. It was shown that over 99% of the simulation results obtained using our machine learning method based on fixed-point and floating-point were consistent when the fractional part was 11-bit. The registers n, ne, and e in Figure 1(b) hold the summed vector sizes for up, upper right, and right, respectively. Registers that hold the summed sizes for each direction and each combination of two directions are omitted in Figure 2(b). The SIZE outputted by the LUT is used as an input by a multiplexer, as shown in Figure 1(b). The multiplexer connects the SIZE to an adder in front of the registers, which holds the summed size for a direction and its combinations, including the direction indicated by the DIRE. The SIZE is added to the current values in each register. The times when the registers capture the added values are controlled by COUNT. The registers capture the added values when an integrated motion vector is input and its size is added to the current values in the registers. After all the motion vectors in an area have been added, the values in the registers are captured by pipeline registers, e.g., pipeline_n in Figure 1(c). All of the outputs of the pipeline registers are connected to the normalization architecture.

2.2 Normalization

The summed sizes for each direction and the combinations of two directions for an area, which are calculated by feature extraction, are normalized in the range [0, +1]. To normalize the values, the maximum value is determined among the summed sizes for one direction and the summed sizes are divided by the maximum value. The summed sizes of the combinations are normalized in a similar manner.

Figure 2 shows a block diagram of the architecture used for normalization. The summed size of each direction obtained by feature extraction is used as an input for a multiplexer, as shown in Figure 2(a), and the summed sizes for the combina-

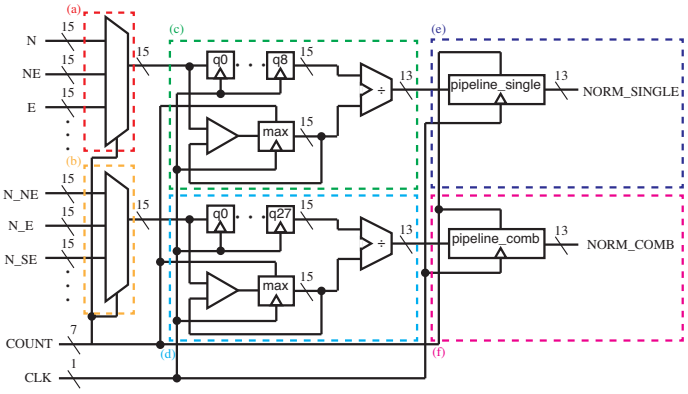

Figure 2: Block diagram of the normalization architecture

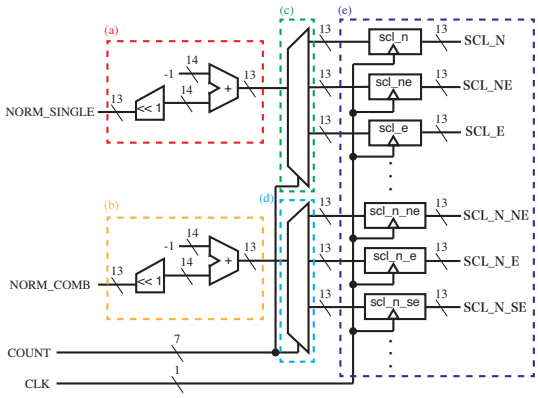

Figure 3: Block diagram of the scaling architecture

tions obtained by feature extraction are also used as inputs by a multiplexer, as shown in Figure 2(b). Each multiplexer connects the input to a register q0, as shown in Figures 2(c) and (d). At the same time, the input value and the current value in the register max (Figures 2(c) and (d)) are compared and the highest value is captured by the register max. This determines the maximum value. q0 and q8 in Figure 2(c), and q0 and q27 in Figure 2(d) are registers used for delay until the maximum value is found. q1 to q7 are omitted from Figure 2(c) and q1 to q26 are omitted from Figure 2(d). After the maximum value is found and captured by the register max, a value in register q8 is divided by the value in the register max (Figure 2(c)). Similarly, the value in register q27 is divided the value in the register max (Figure 2(d)). Values that have been normalized using this method are entered into pipeline registers, i.e., pipeline_single in Figure 2(e) and pipeline_comb in Figure 2(f). The outputs of each register are then used as inputs by a scaling architecture.

2.3 Scaling

When a support vector machine is used, it is important that the input data are scaled [10]. We scale the input data in the range [-1, +1]. The capacity for generalization is greatly improved by scaling the input data in this range. The input data only need to be multiplied by 2 and then 1 is subtracted. This is because the input data have already been normalized to the range [0, +1].

Figure 3 shows a block diagram of the scaling architecture. The normalized input data are used as inputs for a bit shift arithmetic unit (Figures 3(a) and (b)) where they are shifted 1 bit to the left to multiply the input data by 2. The value multiplied by 2 is then submitted as an input to an adder (Figure 3(a) and (b)). The input in the adder is corrected by subtracting 1 from the value. The scaled value is then submitted as an input to a multiplexer (Figures 3(c) and (d)). Each multiplexer is controlled by the COUNT and the input of multiplexer is connected to a suitable pipeline register (Figure 3(e)) at an appropriate time. For example, the input of the multiplexer in Figure 3(c) is connected to a pipeline register called scl_n when the summed size of the up direction has been normalized and scaled. All of the outputs from the pipeline registers are provided as inputs to the simple perceptron architecture.

2.4 Simple perceptron

A simple perceptron is a type of neural network. The perceptron employed in our method outputs a value based on multiple input values entered in the input layer. The input values are multiplied by the synaptic weights and the sum of these values is calculated. If the sum is a positive value, the simple perceptron outputs 1; otherwise, the output is 0. Images are classified into two classes based on the output. The simple perceptron compare its output with the supervised data. If the outputs are different, the simple perceptron updates its synaptic weights to learn the difference. However, the generalization capacity is not improved sufficiently using this method. Thus, we use a support vector machine with the same linear classifier as the simple perceptron. This support vector machine has already learned the data and its synaptic weights are used by the simple perceptron, which improves the generalization capacity to an adequate level. In this study, we used a support vector machine program called LIBLINEAR.

Figure 4 shows the architecture of the simple perceptron. The multiplexer in Figure 4(a) receives the scaled values and connects the inputs to an output one by one. The output of the multiplexer is connected to the multiplier (Figure 4(b)). Another multiplier input is connected to the output of SRAM, which stores the synaptic weight values obtained by LIBLINEAR. Thus, the scaling values are multiplied by the synaptic weights. The output of the multiplier is connected to an adder and another adder input is connected to a register called sum. The values multiplied by the synaptic weights are used as in-

Table 1: FPGA implementation summary (10 × 10, 8-bit, Altera Stratix II)

	LUT's	Registers	Block Memory (byte)	Latency (clk)	Fmax (MHz)
Motion vector estimation	670	790	780	100	80
Machine Learning	3059	2271	416	61	12
Total	3729	3061	1196		

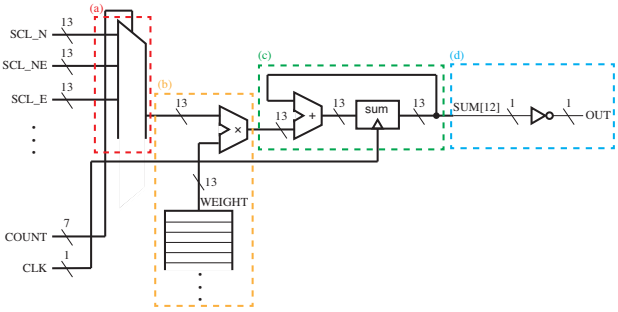

Figure 4: Block diagram of the architecture of the simple perceptron

puts for the adder and they are added to the current value in the register to calculate a sum. If the sum is a positive value, the simple perceptron output is 1; otherwise, the output is 0. A signal with an inverted sign bit in the register's output is obtained as the output of the architecture, as shown in Figure 4(d).

3. FPGA implementation summary

Table 1 shows a summary of our implementation in an FPGA of the machine learning architecture and the motion vector estimation architecture, which we proposed previously [4]. In this implementation, we assumed that the input images had a resolution of 10 × 10 pixels and the depth was 8-bit. We used a commercial FPGA board (MU-200SX II with Altera Stratix II) for the implementation. The Fmax of the machine learning architecture was 12 MHz (Table 1). We assumed that this was due to the divider in the normalization architecture. An image sensor was used to capture images at a resolution of 200 × 200 pixels at 1000 fps, and the images were transmitted directly to an image processor. Therefore, the Fmax must have been more than 32.6 MHz. Thus, the divider must be improved to increase the Fmax of the machine learning architecture.

4. Conclusions

We successfully implemented the proposed machine learning architecture in an FPGA. Therefore, a base of image processing LSI chip which we proposed was completed.

Acknowledgment

We would thank the Semiconductor Technology Academic Research Center (STARC), Japan, for funding this research project.

References

[1] T. Onoye, "Recent trends on media processors for embedded systems," The Journal of The Institute of Image Information and Television Engineers, Vol. 63, No. 9, pp. 1185–1187, 2009.

[2] R. Sale, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Lemieux, P. Pande, C. Grecu, and A. Ivanov, "System-on-chip: reuse and integration," Proc. IEEE, vol. 94, No. 6, pp. 1050–1069, 2006.

[3] P. Garrou, R. Ramm, and C. Bower, "Handbook of 3D Integration: Technology and Applications of 3D Integrated Circuits," 2008.

[4] M. Mori, T. Itou, M. Ikebe, T. Asai, T. Kuroda, and M. Motomura, "FPGA-based design for motion vector estimation exploiting high-speed imaging and its application to motion classification with neural networks," Journal of Signal Processing, Vol. 18, No. 4, pp. 165–168, 2014.

[5] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," Psychol. Rev., 65, pp. 386–408, 1958.

[6] M. L. Minsky and S. A. Papert, "Perceptrons," The MIT Press, 1969.

[7] M. Risenhumber and T. Poggio, "Hierarchical models of object recognition in cortex," Nature Neurosci, 2(11), pp. 1019–1025, 1999.

[8] M. J. Tarr, "News on views: Pandemonium revisited," Nature Neurosci, 2, 932-935, 1999.

[9] V. N. Vapnik, "The Nature of Statistical Learning Theory," Springer, 1995.

[10] C. Hsu, C. C. Chang and C. J. Lin, "A practical guide to support vector classification," Technical report, 2005.