Paper: Rb*--**-****:**

FPGA Implementation of a Binarized Dual Stream Convolutional Neural Network for Service Robots

Yuma Yoshimoto[*,**] **and Hakaru Tamukoh**[*,***]

[*] Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology, Japan
[**] Research Fellow of Japan Society for the Promotion of Science, Japan
[***] Research Center for Neuromorphic AI Hardware, Kyushu Institute of Technology, Japan
Email: yoshimoto.yuma276@mail.kyutech.jp, tamukoh@brain.kyutech.ac.jp

[Received 00/00/00; accepted 00/00/00]

In this study, with the aim of installing an object recognition algorithm on the hardware device of a service robot, we propose a Binarized Dual Stream VGG-16 (BDS-VGG16) network model to realize high-speed computations and low power consumption. The BDS-VGG16 model has improved in terms of the object recognition accuracy by using not only RGB images but also depth images. It achieved a 99.3% accuracy in tests using an RGB-D object dataset. We have also confirmed that the proposed model can be installed in a field-programmable gate array (FPGA). We have further installed BDS-VGG16 Tiny, a small BDS-VGG16 model in XCZU9EG, a System on a Chip with a CPU and a middle-scale FPGA on a single chip that can be installed in robots. We have also integrated the BDS-VGG16 Tiny with a robot operating system. As a result, the BDS-VGG16 Tiny installed in the XCZU9EG FPGA realizes approximately 1.9-times more computations than the one installed in the graphics processing unit (GPU) with a power efficiency approximately 8-times lower than that installed in the GPU.

Keywords: convolutional neural network, depth image, dual stream, field programmable gate array, object recognition

1. Introduction

In recent years, as the labor force has continuously declined within an aging society with fewer children, service robots have been drawing attention. A service robot refers to a robot that supports human work by acting in the same way as humans in environments such as households and stores [1–6]. Such robots are expected to fulfill tasks such as cleaning a child's room or serving as a waiter in a restaurant. To fulfill such tasks, the robot must be able to recognize objects [2, 7], for which the object recognition system is crucial.

In operating a service robot in actual environments, it is necessary to be able to recognize objects with high accuracy, high speed, and low power consumption. Failure in recognizing objects could lead to a failure in fulfilling the assigned task, and low-speed processing could lead to a decline its service quality. A low power consumption is also required for service robots because they are battery driven.

Since AlexNet won the ImageNet Large Scale Visual Recognition (ILSVRC) contest in 2012 [8], convolutional neural networks (CNNs) [9] have become mainstream of object recognition. For example, VGG-16 [10], a CNN with a simple structure, achieved an 8.8% top-5 error rate using the ImageNet 2014 Dataset [11], an object recognition dataset consisting of 1,000 categories.

Inputs to general CNNs, including VGG-16, are only RGB images. On the other hand, many service robots can capture not only RGB images but also depth images with an RGB-D camera. A depth image refers to data with distance information from the camera to the object surface contained in the pixels. Eitel et al. demonstrated that the use of depth images in addition to RGB images in recognizing objects can improve their recognition accuracy [12].

Based on the idea of Eitel et al., we proposed a VGG-16 with a dual-stream structure, i.e., Dual Stream VGG-16 (DS-VGG16) [13]. DS-VGG16 has an RGB stream to learn RGB images and a depth stream to learn depth images. DS-VGG16 achieved a 99.9% accuracy in tests using an RGB-D Object Dataset [14]. By contrast, the said network model with a large-scale structure has the following problems: a large number of computations, a high power consumption, and a long processing time.

Hardware acceleration is effective for improving the object recognition speed. In general, a graphical processing unit (GPU) is used to make a high-speed CNN, which tends to consume a large amount of power, making it unsuitable for service robots [3].

A field-programmable gate array (FPGA) provides an alternative means allowing a GPU to make a high-speed CNN. Nakahara et al. reported that CNNs installed in FPGAs have achieved a higher power efficiency than those installed in a central processing unit (CPU) and a GPU [15]. From the perspective of such high power efficiency and low power consumption, it would be most suitable for a service robot to have a CNN installed in an FPGA [3, 16–18].

In [19], a Binarized Neural Network (BNN) is proposed as an effective means to install a deep neural network (DNN) in hardware. Their proposed method can replace multiplications with XNOR operations by binary-quantizing the weight parameters and activation function outputs at the time of forward propagation.

In the above-mentioned context, this study aims to realize an object recognition system with high accuracy, high speed, and low power consumption. In this paper, we propose a Binarized Dual Stream VGG-16 (BDS-VGG16) [20, 21], a binary DS-VGG16, and a method to install it in an FPGA. We also installed a BDS-VGG16 Tiny, a small version of the proposed model in an FPGA and connected it to the robot operating system (ROS), middleware for robots [22]. In this study, we installed the proposed network in an XCVU190[a], a large-scale FPGA, and in an XCZU9EG[b], a system on a chip (SoC) with a CPU, as well as a mid-scale FPGA integrated on a single chip to enable its installation in a robot. We also verified the operating speed and power efficiency of the system when connected to a robot. In the experiments, BDS-VGG16 achieved a 99.3% accuracy in the evaluations using an RGB-D Object Dataset, and was proved to be installable in an XCVU190. We have further installed BDS-VGG16 Tiny in an XCZU9EG and found that it can operate at a speed approximately 4.7-times higher than that installed in a CPU and approximately 1.9-times higher than that installed in a GPU, and that its power efficiency is approximately 20-times better than that installed in a CPU and approximately 8-times better than that installed in a GPU. Finally, we connected the BDS-VGG16 Tiny installed in the XCZU9EG to an ROS to find that it can process in real time.

This paper consists of six sections. The first section describes the context of this study and its overview. The second section presents studies related to service robots, methods for preprocessing depth images, a CNN, and an FPGA. The third section proposes a method for encoding depth images, a BDS-VGG16 model, and its hardware architecture. Section describes the experiments conducted to measure the BDS-VGG16 object recognition accuracy and the FPGA resource usage rate. The fifth section compares the experimental results of BDS-VGG16 Tiny, a miniaturized BDS-VGG16, a CPU, a GPU, and an FPGA in terms of their processing speeds and power efficiencies. Section 4 also evaluates the system in which the proposed method is connected to an ROS. The sixth section concludes this paper.

2. Related Studies

2.1. Service Robots

Service robots support human tasks in environments such as households and stores. **Figure 1** shows the service robot "Exi@" developed by our team Hibikino-Musashi@Home [7]. Exi@ has a variety of sensors, actuators, and computing devices installed. An RGB-D camera, one of its installed sensors, can capture both RGB and depth images. The computing device installed on the robot is an FPGA.

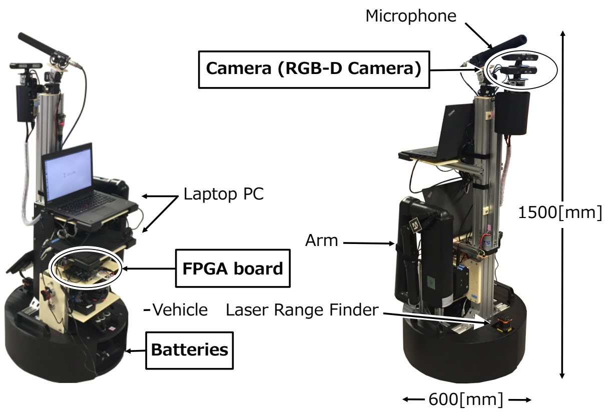


Fig. 1. Service robot "Exi@"

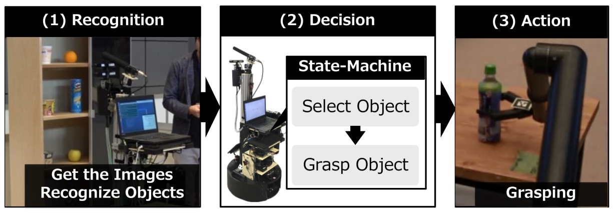


Fig. 2. Robot action flow

Figure 2 shows a flowchart of the robot system. The service robot operates according to the following procedures. (1) Perception: Receive sensor data and recognize its surrounding environment through processing such as object recognition. (2) Decision: The robot decides the actions it should take next on the environmental information it has recognized. (3) Control: The robot acts by controlling the actuator based on the actions it has chosen.

As we can see from the above-mentioned procedures, perception processing is positioned upstream in the robot's action flow. Because it affects every action of the robot, perception processing is crucially important. Therefore, to realize a service robot, it is essential to improve the accuracy and speed of its object recognition system. Because a service robot is battery-driven, we also need to consider the power consumption of its object recognition system.

2.2. ROS

ROS [22] is a middleware for robots. Middleware, lying between the operating system and applications, provides the function of data communication/management/debagging. Above all, ROS has a large number of users globally [c] and has been adopted by many research institutes and businesses [23].

Because various types of algorithms required for the

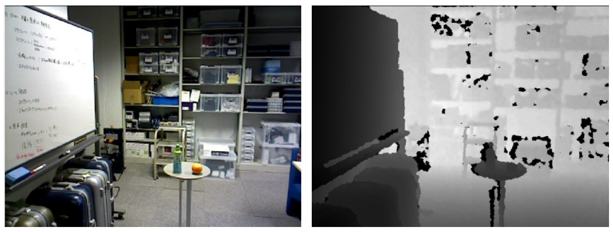


Fig. 3. RGB and depth images

robot system operations are installed as ROS nodes, the robot system operates using the coordinated actions of the nodes. In previously proposed systems with high-speed algorithms developed through their hardware, such algorithms are also designed to act as nodes [3]. Robot systems designed in such a way have the following advantages: No stoppage of any one of the algorithms will stop the robot instantaneously, it is quite easy to transfer from a conventionally used algorithm to a new algorithm, and the algorithms have a high reusability.

In this study, we installed the proposed object recognition system as an ROS node and connected it to a robot system with an ROS.

2.3. RGB and Depth Images

An RGB-D camera is installed on the service robot. **Figure 3** shows the (a) RGB image and (b) depth image captured by the RGB-D camera. In the RGB image in **Figure 3** (a), a whiteboard, a round table, and shelves are shown. The pixels of the depth image represent the distances from the camera to the object surfaces. In **Figure 3** (b), the whiteboard and round table, both positioned close to the camera, are represented in black because their distances from the camera are nearly zero. By contrast, the shelves, positioned at a long distance from the camera, are represented in white because their distances have large values.

The features of an RGB image contain the appearance information of an object such as the color and patterns. The weak points of an RGB image include its direct susceptibility to the effects of the illumination conditions. By contrast, as the features of the depth image, it contains the shape information of an object: Its strong points are its robustness to the illumination conditions, and its weak points are its likelihood to contain some missing values (noises), as shown in the black spots in **Figure 3** (b).

2.4. How to Remove Noise from Depth Image

As described above, a depth image contains noise. A depth sensor calculates the distance to an object on the reflected waves of the infrared ray and laser radiated to the object surface, and thus it may fail to measure the distance to an object owing to the effects of the surface material of the object. In the event of a failure in the distance measurements, such distance points constitute missing values and appear as noise. Such noises are often represented in the data as NaN or 0, and in the RGB-D object dataset, in particular, which is stored as image files, noises are represented as 0. As a method for repairing such noises, Lai et al. proposed the application of a recursive median filter (RMF) to the pixels that contain noise [14].

Figure 4 shows the depth images of an apple as captured by the camera: (a) an unrepaired depth image and (b) an RMF-applied depth image, in which the only brightness has been adjusted for easier viewing. We can see from image (b) that the noise appearing as black spots in image (a) disappeared by applying the RMF. Now, we normalize the images by linearly transforming them into a value range of 0 to 255. **Figure 4** (c) and (d) show the normalized images of Figure (a) and (b), respectively. **Figure 4** (e) and (f) show the histograms of Figure (c) and (d), respectively. We can see from **Figure 4** (f) that the values are dispersed in the area between 0 and approximately 255, and from **Figure 4** (e) that the noises constitute outliers and that the values are concentrated in the area between approximately 150 and 255, except for the noise indicated by a 0, which seems to suggest that the value range between approximately 150 and 255 is practically the only range we can use to represent the object shape. In other words, removing noise from an image seems to increase the value range of the normalized image, which we can practically use.

2.5. Encoding Methods for Depth Images

To appropriately extract features from depth images, several methods have been proposed to encode such images before the CNN learns them. The proposed encoding methods expand a one-channel depth image to a three-channel image. **Figure 5** shows the depth images encoded by these methods. These encoding methods have the following features

(a) Surface normal method: Bo et al. proposed a surface normal method [24]. **Figure 5** (a) shows an image encoded using the surface normal method. The surface normal method calculates the surface normal of each pixel of a depth image and directly substitutes the vector components x, y, and z in three channels.

(b) HHA Method: The horizontal disparity, height above the ground, and angle (HHA) method is an encoding approach proposed by Gupta et al. [25]. The HHA method seeks a horizontal disparity, height above ground, and angle toward the vertical direction and stores them in three channels. **Figure 5** (b) shows a depth image encoded by the HHA method. The HHA method, which combines environmental variables with depth imaging, should be suited to scene images rather than images that only contain objects.

(c) ColorJET Method: The ColorJET method is the method proposed by Eitel et al. [12]. It processes in the following order: (1) linear transform (normalize) a depth image such that its values reach the range

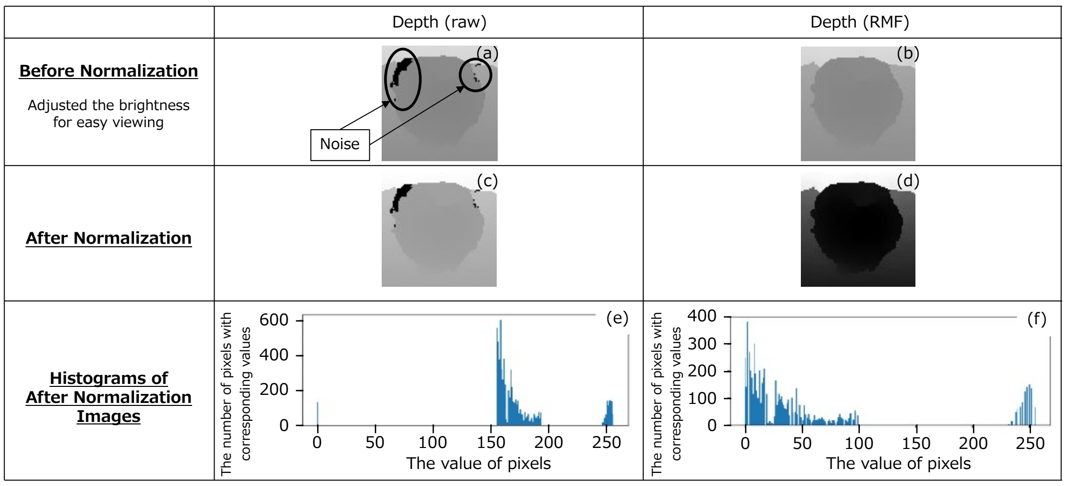


Fig. 4. Comparison of unrepaired depth image with RMF-applied depth image

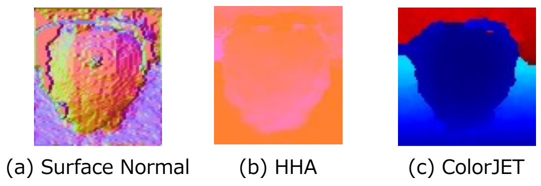


Fig. 5. Comparison of depth image encoding methods

of 0 to 255, and (2) map the normalized image on a JET color map. In the above-mentioned processing, a minimum value is allocated to blue, a median value to green, and a maximum value to red. **Figure 5** (c) shows the image encoded using the ColorJet method.

In the dual-stream type network proposed by Eitel et al., it is most appropriate to use the images encoded by the ColorJET method rather than those encoded by the surface normal method or the HHA method [12]. To improve the performance of dual stream type network models, this study uses the ColorJET method to encode the depth images.

2.6. Dual Stream Type Networks

2.6.1. Model Proposed by Eitel et al.

The model proposed by Eitel et al. uses both RGB images and depth images to recognize objects [12]. The model is composed of an RGB stream to learn RGB images, a depth stream to learn depth images, and a section to integrate both stream outputs and function as a classifier. Each stream, consisting of five convolutional layers, has a CaffeNet-based structure [26]. Eitel et al. used the ColorJET method as a preprocessing method for inputting the depth images. Eitel et al. reported that the use of the above-mentioned method has made their model more accurate in comparison with those that use only RGB images.

2.6.2. Dual Stream VGG-16

Dual Stream VGG-16 (DS-VGG16) combines VGG-16, a highly accurate object recognition CNN, with the model proposed by Eitel et al. [13]. **Figure 6** shows the structure of DS-VGG16. This method achieves a 99.9% accuracy in tests using the RGB-D Object Dataset published by Washington University [14]. Similar to the model proposed by Eitel et al., this model is composed of (a) an RGB stream for RGB images, (b) a depth stream for depth images, and (c) an integration section, in which the outputs of both streams are integrated. Each stream has a structure for VGG-16 instead of CaffeNet.

2.7. FPGA

The FPGA is a large-scale integration circuit reconfigurable by users, and is configured to allow building an arbitrary digital circuit by place-and-routing a look-up table, a flip flop, a digital signal processor, and random access memory (RAM).

An FPGA, which can be efficiently installed to realize the desired functions, can suppress wasteful energy consumption as compared to a GPU, thus suppressing the power consumption and heating. Another computation source that can suppress the power consumption and heating as compared to a GPU is an application specific integrated circuit (ASIC), which has a disadvantage in that, once produced, cannot be modified, making it obsolete for use in robot applications for which new algorithms are applied every day. An FPGA, the circuit of which is rewritable and more suitable for robot applications than ASIC, is widely used as a robot computing device [3, 16–18, 27, 28].

2.8. Binarized Neural Network

A Binarized Neural Network (BNN) is a quantification method effective for installing neural network hardware, as proposed by Hubara et al. [19]. Whereas conventional neural networks are installed using floating-point numbers, such floating-point arithmetic requires

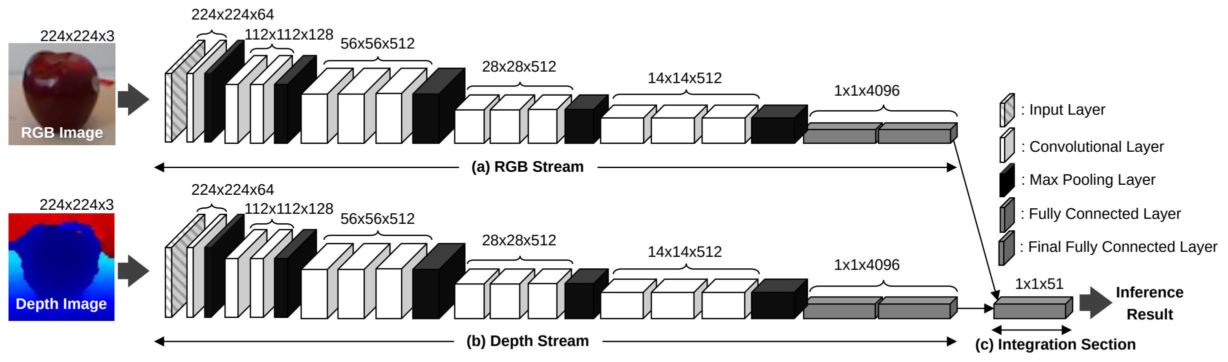


Fig. 6. Structure of DS-VGG16

Table 1. Comparison between BNN and Conventional Neural Networks

	BNN	Neural Network
Weight representation	Binary Number $W_{bi} = Sign(W_i)$	Floating Point Number
The sum of neurons outputs of each layer	$u = \sum_{i=1}^{n} x_{bi} W_{bi} + b$ Replaceable with XNOR	$u = \sum_{i=1}^{n} x_i W_i + b$ Required Multiplication
Activation Function Output	Binary Number $y = Sign(u)$	Floating Point Number $y = f(u)$

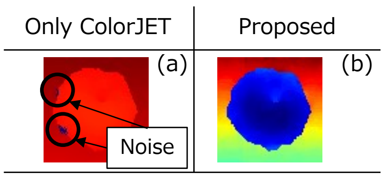


Fig. 7. Comparison between (a) depth image applied with ColorJET only and (b) depth image applied with RMF as well

enormous hardware resources. To cope with the above-mentioned problem, a BNN realizes a neural network with fewer hardware resources than floating-point numbers by binary-quantifying the floating-point numbers.

Table 1 shows the differences between the conventional neural network and a BNN. In a BNN, the weight parameters and activation function outputs at the time of a forward propagation are made binary using values of 1 and -1. Hubara et al. concluded that a BNN has nearly the same accuracy as a 32-bit neural network.

In realizing a BNN with hardware, the use of 1s and 0s instead of 1s and -1s enables us to reduce the bit width of variables as well as replace multiplications with XNOR operations. **Table 2** shows the multiplications using values of 1 and -1, which require two bits. On the other hand, **Table 3** shows the XNOR operations using 1s and 0s. Because **Table 2** and **3** have the same structure in which the -1s in **Table 2** are replaced with 0s, we may regard a multiplication of 1×-1 in the network as an XNOR operation of 1s and 0s. By so doing, we can install a BNN with values of 1 and -1 in use as XNOR operations of 1s and 0s in the circuit.

Table 2. Multiplication Table

Input A	Input B	Output
-1	-1	1
-1	1	-1
1	-1	-1
1	1	1

Table 3. XNOR truth table

Input A	Input B	Output
0	0	1
0	1	0
1	0	0
1	1	1

3. Proposed Method

3.1. Depth Image Preprocessing Method

This study uses the ColorJET method, which is reportedly an effective method for preprocessing depth images. One of the problems with the ColorJET method is that it is susceptible to missing values (noises). **Figure 7** (a) shows a depth image as preprocessed by applying only the ColorJET method. In **Figure 7** (a), where noise constitutes outliers in the depth image, the value area is shifted toward the minimum and maximum values on the JET color map, and thus the image is represented in the same color in most areas.

In this study, therefore, the application of the RMF prior to the ColorJET method is proposed. **Figure 7** (b) shows a depth image as acquired by applying the proposed method, where noise is removed in advance such that we can allocate most of the value area of the JET color map to objects and the background.

3.2. Binarized Dual Stream VGG-16

In this study, a BDS-VGG16 model is proposed. **Figure 8** shows the model structure of the network. This model is based on DS-VGG16. **Table 4** shows the differences between DS-VGG16 and BDS-VGG16. Both DS-VGG16 and BDS-VGG16 consist of fully connected layers to infer objects from the outputs of both the RGB stream and depth stream. In the BDS-VGG16, its convolutional layer and fully connected layer are made binary by means of the BNN method, as shown in **Table 4**. To stabilize the learning of the proposed network model, it also has a batch normalization layer inserted between the convolutional layer and the fully connected layer [29].

3.3. Circuit Architecture

Figure 9 shows the units in the circuit architecture proposed in this study and their operation flow. The circuit architecture has an RGB stream to treat RGB images and feature maps, and a depth stream to treat depth images and feature maps. Each unit contains a binarized input layer, binarized convolutional layer unit, binarized max pooling layer unit, binarized fully connected layer unit, and binarized final fully connected layer unit. To save on the circuit area, this circuit architecture uses the same binarized convolutional layer unit for the computations of the convolutional layer by changing the inputs and weights each time. In particular, the binarized input layer unit, binarized convolutional layer unit, binarized fully connected layer unit, and binarized final fully connected layer unit contain a BN layer connected in the later stage of each layer of the network model shown in **Figure 8**. Regarding such BN layers, Yonekawa et al. reported that BN layers can be realized in a BNN through simple additions [30]. Therefore, we adopted their proposed method in our proposed network model. Nakahara et al. improved the circuit into a dual stream based on the architecture proposed by Yonekawa et al. [30, 31].

3.3.1. Binarized Input Layer Unit

Figure 10 shows the binarized input layer unit. The unit performs the convolutional operations of the input layer. In **Figure 10**, $x_{c,ch}$ and $x_{d,ch}$ denote the ROIs and channel ordinal numbers as cut out of the input RGB image and the depth image, respectively, and $w_{c,ch}$ and $w_{d,ch}$ denote the weights and channel ordinal numbers of the RGB stream and the depth stream, respectively. Here, CH denotes the number of input channels and the number of filters, which are fixed at 3. In addition, K denotes the filter size, which is also fixed at 3. Finally, B denotes a BN value, and f_c and f_d denote the output feature maps corresponding to the ROIs in the current attention.

With the three input channels being integers of 0 to 255 and $w_{c,ch}$ and $w_{d,ch}$ being binary, this circuit can realize multiplications of $x_{c,ch} \times w_{c,ch}$ and $x_{d,ch} \times w_{d,ch}$ with the selector. The adder tree in the later stage seeks a total sum of the multiplications of $x_{c,ch} \times w_{c,ch}$ and $x_{d,ch} \times w_{d,ch}$. The BN values are added at the end.

The ROI is input into this circuit by sliding it over the input image. The circuit also operates as pipelining. The above-mentioned operations are processed in parallel for RGB images and depth images.

3.3.2. Binarized Convolutional Layer Unit

Figure 11 shows the binarized convolutional layer unit. In **Figure 11**, $f_{c,ch,n}$, and, $f_{d,ch,n}$ denote the ROIs in the nth channel as cut out of the feature maps of the RGB stream and the depth stream, respectively; in addition, $w_{c,ch,n}$ and $w_{d,ch,n}$ denote the weights in the nth channel of the RGB stream and the depth stream, respectively. In this circuit, K is fixed at 3, and CH varies with the layers.

With the inputs to this unit $f_{c,ch,n}$, and $f_{d,ch,n}$ being both binary, they can be computed at the XNOR gate. In this circuit, the multiplications of $f_{c,ch,n} \times w_{c,ch,n}$ and $f_{d,ch,n} \times w_{d,ch,n}$ are conducted in parallel using as many XNOR gates as the number of CHs in each stream.

The adder tree seeks the total sum of the multiplications of $f_{c,ch,n} \times w_{c,ch,n}$ and $f_{d,ch,n} \times w_{d,ch,n}$. Whereas feature maps and weights are represented by 0s and 1s in this circuit, a constant is added in the comparator to compare them with the reference value of 0. In this circuit, a constant and a BN value are added simultaneously.

The above-mentioned operations are processed in parallel for both RGB and depth images.

3.3.3. Binarized Max Pooling Layer Unit

Figure 12 shows the binarized max pooling layer unit. In this circuit, K is fixed at 2, and ch denotes the channel to be processed.

With the input values to this unit being 0s or 1s, max pooling can be represented using an OR gate. The OR operations are processed in parallel for both RGB and depth images.

3.3.4. Binarized Fully Connected Layer Unit

Figure 13 shows the binarized fully connected layer unit. In the circuit diagram, $f_{c,n}$ and $f_{d,n}$ denote the features of the RGB and depth streams, respectively; $w_{c,n}$ and $w_{d,n}$ denote their weights. Each stream has N features, and n is an index for them.

As in other units, the RGB and depth streams are simultaneously processed in this unit.

3.3.5. Binarized Final Fully Connected Layer Unit

Figure 14 shows the binarized final fully connected layer unit. This unit represents the integration section (**Figure 8** (c)), where the RGB stream and the depth stream are integrated together. In **Figure 14**, input f denotes an integration of f_c and f_d, and y denotes the output of the unit.

In this unit, outputs are not made binary and their computation values are treated as class probabilities.

4. Experiments

4.1. Datasets setup

This study evaluates the recognition accuracies of network models using the RGB-D Object Dataset [14]. **Figure 15** shows a part of the dataset. This dataset contains 51 classes of objects, and household articles such as apples (**Figure 15** (a) (b)), bananas (**Figure 15** (c) (d)), and coffee mugs (**Figure 15** (e) (f)) were selected as objects. The dataset consists of 207,920 pairs of data, each pair of which consists of RGB images and their corresponding depth images. **Figure 15** (a), (c), and (e) shows examples

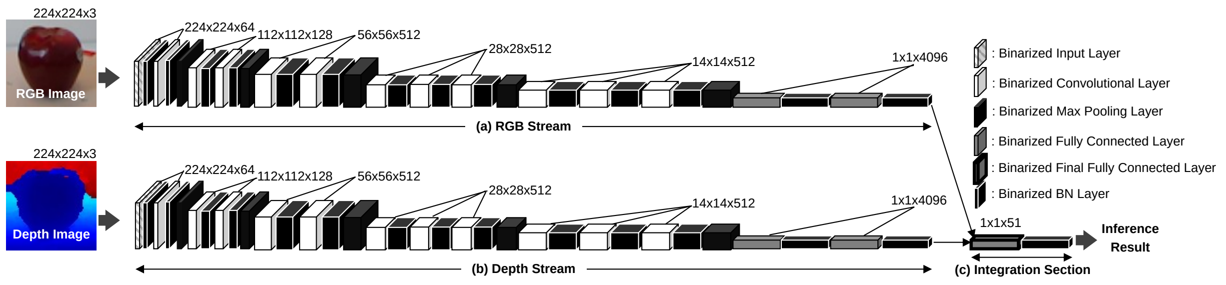


Fig. 8. Structure of BDS-VGG16

Table 4. Differences between DS-VGG16 and BDS-VGG16

DS-VGG16	BDS-VGG16	Note
Convolutional Layer	Binarized Convolutional Layer	For reduction of weight memory size and the number of multipliers.
Max Pooling Layer	Binarized Max Pooling Layer	Not changed. This layer does not use weights and multipliers.
Fully Connected Layer	Binarized Fully Connected Layer	For reduction of weight memory size and the number of multipliers.
N/A	Batch Normalization Layer	For training stability, BN Layers are inserted.

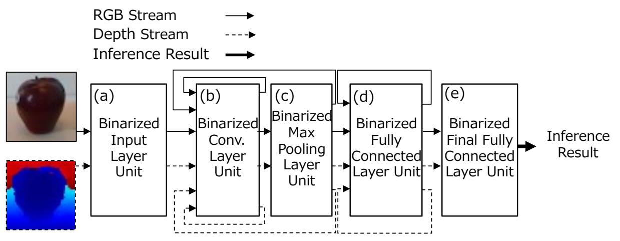

RGB Stream
Depth Stream
Inference Result
(a) Binarized Input Layer Unit
(b) Binarized Conv. Layer Unit
(c) Binarized Max Pooling Layer Unit
(d) Binarized Fully Connected Layer Unit
(e) Binarized Final Fully Connected Layer Unit
Inference Result

Fig. 9. Operation flow of each unit of BDS-VGG16

Table 5. Test accuracies of ColorJET only method and method using both RMF and ColorJET

Methods	Depth
Only ColorJET	58.5%
RMF and ColorJET	61.5%

of RGB images, and **Figure 15** (b), (d), and (f) shows examples of depth images. In this study, we randomly selected 75% of the data pairs from each class to make them learn the data and the remaining 25% of data pairs as test data.

4.2. Evaluate Effectiveness of RMF in Preprocessing Depth Images

Here, we evaluate the effectiveness of applying the RMF and ColorJET methods to depth images for the network learning. We verify the effectiveness by using only the depth images in the RGB-D object dataset described in Subsection 4.1. In the experiments, we provided two binarized VGG-16s and had one of them learn the depth images applied using only the ColorJET method, and had the other one learn the depth images applied using both the RMF and ColorJET methods to compare their post-learning test accuracies.

Table 5 shows the experimental results, from which we can see that the network model that learned the dataset using the ColorJET method after the RMF achieved a better learning accuracy than the model that learned the dataset using only the ColorJET method. Applying both the RMF and ColorJET methods to depth images is more effective in improving the network learning accuracy than applying the ColorJET method only.

4.3. Experiments Evaluating Recognition Accuracy of BDS-VGG16

We experimentally evaluated the recognition accuracy of BDS-VGG16. During the experiments, we evaluated the recognition accuracy using the learning data and test data, as selected in Subsection 4.1. Randomly selected learning data (90%) are used for the network learning, and the remaining data (10%) are used as validation.

Table 6 shows the experimental results. We compared them with the test accuracies of Eitel's model [12], Schwarz's model [32], and DS-VGG16 [13] and found that the accuracy of the proposed BDS-VGG16 remained 0.6% lower than that of DS-VGG16, which seems attributable to the fact that the binary network as a whole has narrowed the value areas of the units in the network so much as to decline its expressive power. However, the proposed network model achieved an accuracy 12.8% higher than Binarized VGG-16, which seems attributable to the effects of learning the depth images as well owing to its dual stream structure. Its accuracy improved by 5.2% compared to Eitel ’s model and by 8.0% compared to the Schwarz model. We can see from the above-mentioned experimental results that the proposed BDS-VGG16 has a higher accuracy than Eitel ’s or Schwarz ’s models.

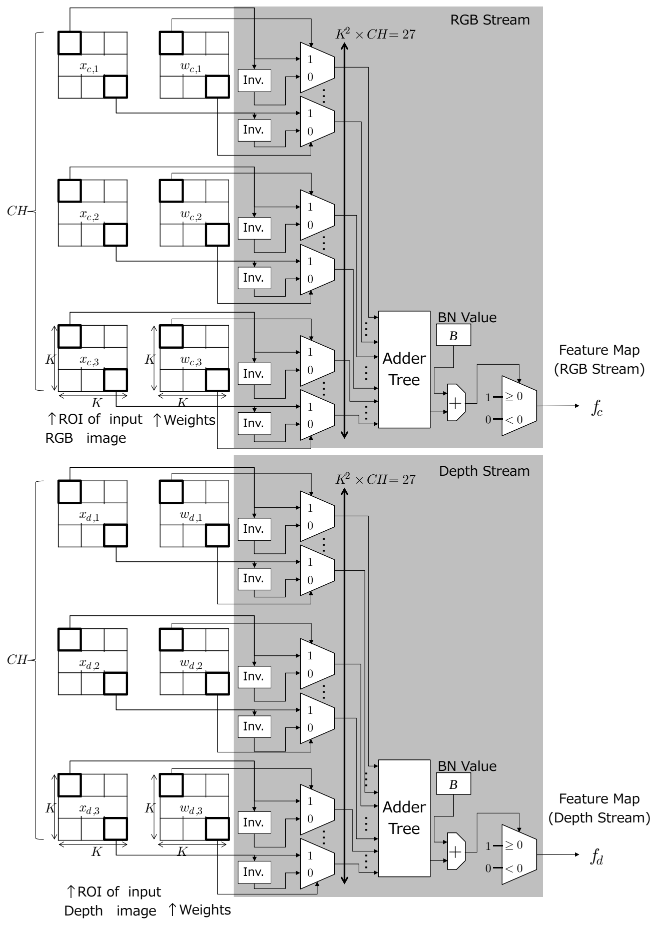


Fig. 10. Binarized Input Layer Unit

Table 6. Comparison of test accuracies between BDS-VGG16 and other methods

	RGB	RGB-D
Eitel *et al.* (2015) [12]	92.1%	94.1%
Schwarz *et al.* (2015) [32]	84.1%	91.3%
Binarized VGG-16	86.5%	–
DS-VGG16 (RMF + ColorJET) (2018) [13]	–	99.9%
BDS-VGG16 (RMF + ColorJET) (Ours)	–	99.3%

4.4. Evaluation of Circuit Resource Usage in Installing FPGA

We evaluated the resources required by the circuit to install an FPGA based on the creation of its circuit architecture. In this study, we created a circuit with a high-level synthesis (HLS) using the Xilinx SDx 2018.3 [d]. The target devices were an XCVU190 and XCZU9EG. The XCVU190 is a relatively large FPGA, and the XCZU9EG is an SoC incorporating a CPU as a processing system (PS) and an FPGA as a programmable logic (PL), allowing the chip to easily enable communication between the FPGA and the robot through the CPU. It is also installed in our service robot, which is under development [7].

Table 7 shows the circuit synthesis results, from which we can see that the proposed circuit can be installed on an FPGA such as an XCVU190.

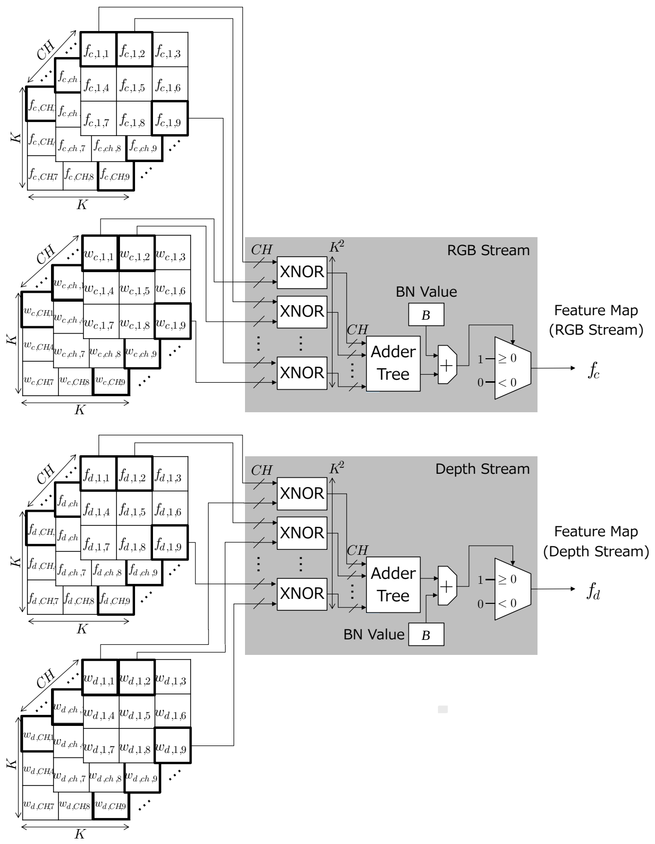


Fig. 11. Binarized Convolutional Layer Unit

Table 7. FPGA Synthesis Results of BDS-VGG16

	BDS-VGG16			
	XCVU190		XCZU9EG	
	Resource Usage	Available	Resource Usage	Available
BRAM	2,589 (68.49%)	3,780	**2,628 (288.16%)**	912
FF	91,797 (4.27%)	2,148,480	159,111 (29.03%)	548,160
LUT	141,548 (13.18%)	1,074,240	176,556 (64.42%)	274,080
DSP	1 (0.06%)	1,800	0 (0%)	2,520

5. Installation of FPGA

We found that our proposed BDS-VGG16 was too large for the FPGA of the XCZU9EG installed in a robot. Therefore, we propose BDS-VGG16 Tiny, whose input image size is smaller and whose channels are fewer than those of a BDS-VGG16. We first checked the network size to determine whether it could be installed on the XCZU9EG. Next, we proposed a network and installed it on an FPGA to evaluate its inference speed and power efficiency, among other factors.

5.1. Check Installable Network Size on XCZU9EG

We checked the network size to determine whether it could be installed on the XCZU9EG. We can see from

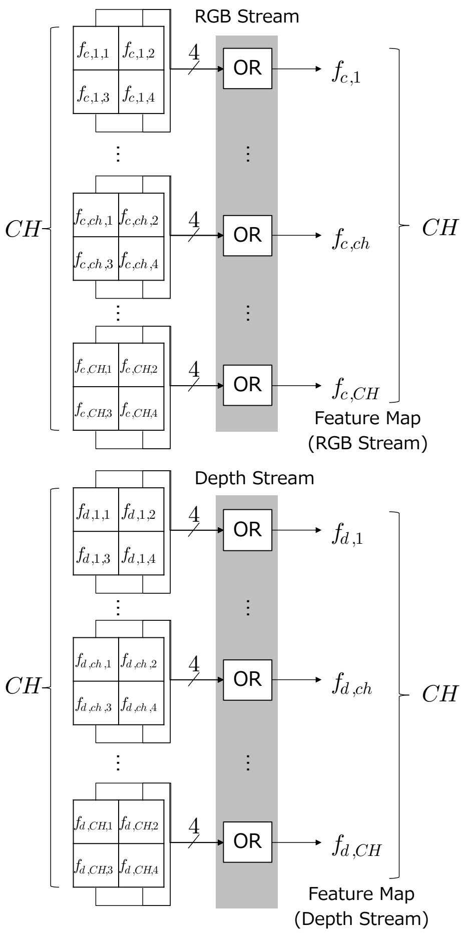


Fig. 12. Binarized Max Pooling Layer Unit

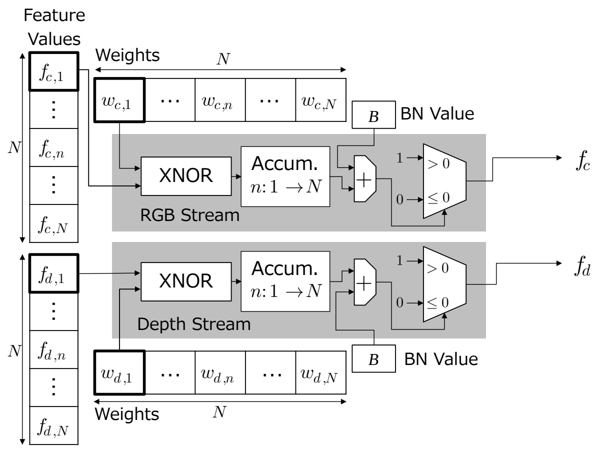


Fig. 13. Binarized Fully Connected Layer Unit

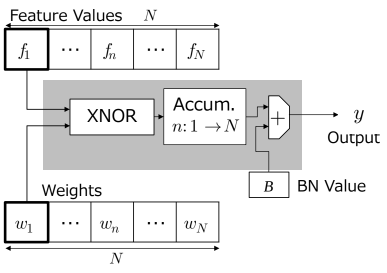


Fig. 14. Binarized Final Fully Connected Layer Unit

Table 8. Summary of BRAM usage required by network

Number of Channel	Input Image Size				
	48×48	96×96	112×112	128×128	224×224
64	30.15%	34.98%	35.80%	36.90%	57.29%
128	53.84%	60.96%	65.57%	68.64%	102%
192	79.77%	89.42%	95.56%	**99.73%**	N/A
256	102%	N/A	N/A	N/A	N/A

Table 7 that the resource shortage of BRAM makes it difficult to install the proposed network on the XCZU9EG. The proposed method uses BRAM to store the weight parameters and input images. Therefore, in this subsection, we check the BRAM resource to be consumed by varying the input image size and the number of channels in the middle layers.

Table 8 shows the BRAM usages required by the network. N/A indicates that no tests have been conducted. We can see from **Table 8** that the network model with an input image with a pixel resolution of 128×128 and 192 channels achieves the highest BRAM usage and that the largest-scale network can be installed on the XCZU9EG as the target FPGA.

5.2. Proposed BDS-VGG16 Tiny

In this study, we propose BDS-VGG16 Tiny based on the results shown in **Table 8**. **Figure 16** shows the proposed network model. The said network has been changed in terms of the following points from BDS-VGG16: (1) the input image size has been changed to a resolution of 128×128, and the data size has become one-quarter the original size, (2) the number of channels has been changed to 192 in all convolutional layers except for the input layer, and (3) as Nakahara et al. have reported, replacing the average pooling layers with fully connected layers except for the final layer in the BNN will produce nearly the same effect [31]. We replaced the binarized fully connected layers prior to the final layer with binarized average pooling layers.

Figure 17 shows the binarized average pooling layer unit to be used in BDS-VGG16 Tiny. In these circuits, average pooling is realized using the adder tree and the selector. The kernel size K has a pixel resolution of 8×8, and the number of channels in the input feature map is 1.

Finally, we propose the circuit configurations for installing an FPGA in BDS-VGG16 Tiny. **Figure 18** shows the operation flows of the units used to realize BDS-VGG16 Tiny.

5.3. High-Level Synthesis Results

We conducted a high-level synthesis of the proposed BDS-VGG16 Tiny with an Xilinx SDx 2018.3 for

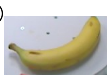
Name	Apple	Banana	Coffee Mug
RGB Images	(a)	(c) 	(e)
Depth Images Adjusted the brightness for easy viewing	(b)	(d)	(f)

Fig. 15. Part of RGB-D Object Dataset

Table 9. Synthesis Results

	BDS-VGG16 tiny	
	XCZU9EG	
	Resource Usage	Available
BRAM	909.50 (99.73%)	912
FF	51,449 (9.39%)	548,160
LUT	76,195 (27.80%)	274,080
DSP	1 (0.04%)	2,520

XCZU9EG as the target device. **Table 9** shows the synthesis results, from which we can see that the proposed model can be installed in an XCZU9EG.

5.4. Evaluation Experiments on Recognition Accuracy of BDS-VGG16 Tiny

We applied BDS-VGG16 Tiny to learn the images and compared its recognition accuracy with those of other binary networks. We conducted the experiments using the dataset set up, as described in Subsection 4.1, in the same way as indicated in Subsection 4.3. **Table 10** shows the experimental results, from which we can see that the accuracy of BDS-VGG16 Tiny has declined by 8.1% compared to that of BDS-VGG16, which seems attributable to the changes in the input image size and in the number of channels. However, the recognition accuracy of the proposed network model has improved by 4.7% compared to that of the binary VGG-16.

5.5. Comparison of Performance of BDS-VGG16 Tiny with other Computing Devices

We compared the inference speed and efficiency of BDS-VGG16 Tiny when installed in other computing devices. First, we installed BDS-VGG16 Tiny in the other computing devices (embedded CPU, CPU, and GPU) and

Table 10. Comparison of accuracies among BDS-VGG16 Tiny and other binary methods

	RGB	RGB-D
Binarized VGG-16	86.5%	–
BDS-VGG16 (RMF + ColorJET) (Ours)	–	99.3%
BDS-VGG16 tiny (RMF+ColorJET) (Ours)	–	91.2%

Table 11. Specifications of laptop PC used in experiments

CPU	Intel Core i7-7850HK
Memory	32GB
GPU	NVIDIA GeForce GTX 1080 8GB
OS	Ubuntu16.04
Language	Python 3.5
Framework	Chainer 1.17.1

measured their inference speeds and power consumption. In measuring the inference speeds and power consumption, we used an ARM Cortex-A53 incorporated in XCZU9EG as an embedded CPU. We used a laptop PC with the specifications shown in **Table 11** to measure the inference speeds and power consumptions of the CPU and GPU. In measuring the inference speeds, we seek the average time of inferring single data 100 times with BDS-VGG16 Tiny installed in the FPGA, embedded CPU, CPU, and GPU. A single dataset contains RGB images and depth images already applied with the RMF and ColorJET methods. The inference speed measurements of BDS-VGG 16 Tiny installed in the FPGA include the communication time between the PS and PL, and those of BDS-VGG16 Tiny installed in the GPU include the time require to transfer data to the GPU memory. We measured the power consumption of BDS-VGG16 Tiny installed in the FPGA and in the embedded CPU by connecting a voltmeter and an ammeter between the FPGA board and the AC adaptor. In measuring the power consumption of BDS-VGG16 Tiny installed in the CPU and GPU, we connected a wattmeter between the AC adaptor and electric outlet of the PC.

Table 12 shows the measurement results. "Frame" in **Table 12** refers to processing an RGB image as well as a depth image. We can see from Table 12 that the inference speed of BDS-VGG16 Tiny installed in the FPGA is approximately 117-times faster than that installed in the embedded CPU, approximately 4.7-times faster than that installed in the CPU, and approximately 1.9-times faster than that installed in the GPU. The efficiency of BDS-VGG16 Tiny installed in the FPGA is approximately 114-times higher than that installed in an embedded CPU, approximately 20-times higher than that installed in the CPU, and about 8-times higher than that installed in the GPU. We can tell from the above-mentioned measurement results that BDS-VGG16 Tiny installed in the FPGA is superior in terms of both speed and efficiency to those installed in the CPU and GPU.

5.6. ROS Installation

In this study, we built an object recognition system by connecting BDS-VGG16 Tiny installed in an FPGA to an ROS.

5.6.1. System Structure

Figure 19 shows the system structure. Each program was installed as a node in the ROS, where the nodes coordinate their operations by publishing and subscribing

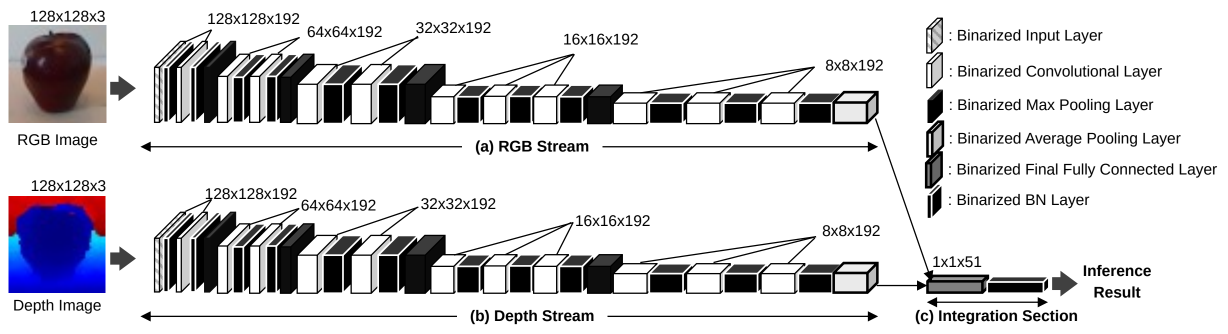


Fig. 16. Structure of BDS-VGG16 Tiny

Table 12. Comparison of recognition speeds and power efficiency

	ZCU 102			LaptopPC (AlienWare)		
	This Research (ZCU102) XCZU9EG	Embedded-CPU (ARM Cortex-A53)	Idling	CPU (Core i7-7850HK)	GPU (Nvidia GTX 1080)	Idling
Inference Speed [msec/Frame] (FPS)	12.5 (80.0)	1,456.9 (0.69)	N/A	58.8 (17.0)	23.3 (42.9)	N/A
Power Consumption [W]	21.3	20.7	20.5	89.2	92.4	32
Efficiency [FPS/W]	3.756	0.033	N/A	0.191	0.464	N/A

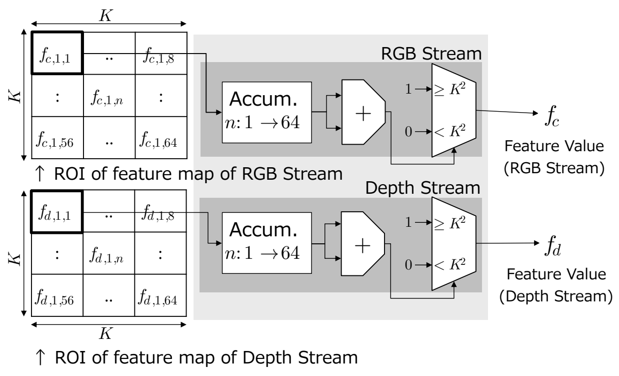


Fig. 17. Binarized Average Pooling Layer Unit

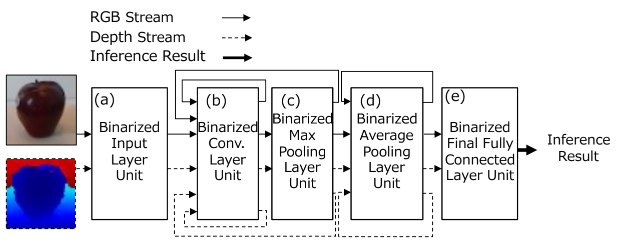

RGB Stream
Depth Stream
Inference Result
(a) Binarized Input Layer Unit
(b) Binarized Conv. Layer Unit
(c) Binarized Max Pooling Layer Unit
(d) Binarized Average Pooling Layer Unit
(e) Binarized Final Fully Connected Layer Unit
Inference Result

Fig. 18. Operation flows of units of BDS-VGG16 Tiny

data, called topics. The operation flow of the system is as follows: (1) An Intel RealSense (RGB-D camera) [e] is used to photograph the objects. (2) The realsense_driver node receives the RGB images and depth images from the RealSense and publishes them as respective topics. (3) The crop_imgs node subscribes the RGB images and depth images and trims and publishes them. (4) The recursive_median_filter node subscribes the depth images, removes their noises with the RMF, and publishes noiseless depth images. (5) The pre_pro node subscribes the RGB images and depth images, resizes both of them, and publishes both images after applying ColorJET to the depth images. (6) The inference_hw node subscribes both images and sends them to the FPGA through socket communications. (7) The PS unit installed in the PS of the FPGA receives the RGB images and depth images and sends them to the PL installed in BDS-VGG16 Tiny. (8) BDS-VGG16 Tiny installed in the PL of the FPGA infers the object classes from their RGB and depth images. (9) The PS unit installed in the PS of the FPGA receives the inference results and transmits them to the inference_hw node through socket communications. Finally, (10) the inference_hw node publishes the inference results.

5.6.2. System Evaluation Experiments

We experimentally verified that the system operates in real time. During the experiments, we checked the system's operation period by measuring the output cycle of the Results Topic when RealSense operates at a specified 30 [fps]. To check whether RealSense operates at 30 [fps], we checked the operation period of the realsense_driver node by measuring the output cycle of the original RGB Image Topic as well. We measured the output cycle 10 times in a row to determine its average and dispersion. We used a laptop PC having the specifications shown in **Table 11** as the operating environment for the ROS.

The experimental results show that the output cycle of the Results Topic is 24.321 [fps] on average and 1.1851×10^{-2}[fps^2] in terms of dispersion when the output cycle of the original RGB image topic is 29.912 [fps] on average and 1.4609×10^{-4}[fps^2] in terms of dispersion. We can see from the above-mentioned experimental results that the proposed system can conduct processing operations in real time.

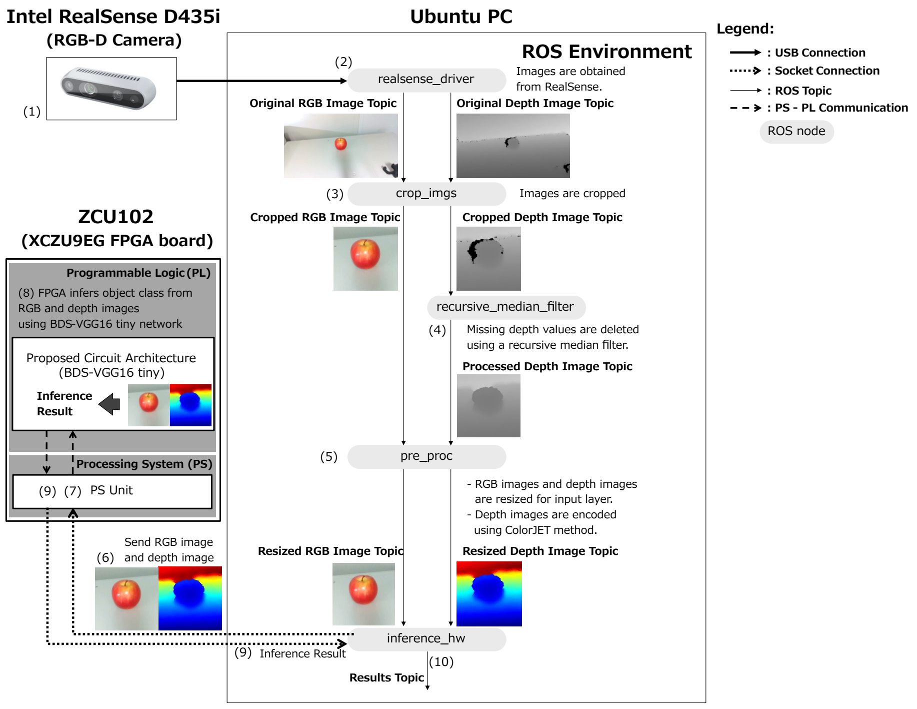

Intel RealSense D435i
(RGB-D Camera)
(1)
Ubuntu PC
ROS Environment
(2)
realsense_driver
Images are obtained from RealSense.
Original RGB Image Topic
Original Depth Image Topic
(3)
crop_imgs
Images are cropped
Cropped RGB Image Topic
Cropped Depth Image Topic
recursive_median_filter
(4)
Missing depth values are deleted using a recursive median filter.
Processed Depth Image Topic
(5)
pre_proc
- RGB images and depth images are resized for input layer.
- Depth images are encoded using ColorJET method.
Resized RGB Image Topic
Resized Depth Image Topic
inference_hw
(9) Inference Result
(10)
Results Topic
ZCU102
(XCZU9EG FPGA board)
Programmable Logic (PL)
(8) FPGA infers object class from RGB and depth images using BDS-VGG16 tiny network
Proposed Circuit Architecture (BDS-VGG16 tiny)
Inference Result
Processing System (PS)
(9) (7) PS Unit
(6) Send RGB image and depth image
Legend:
: USB Connection
: Socket Connection
: ROS Topic
: PS - PL Communication
ROS node

Fig. 19. Object recognition system installed in ROS

6. Conclusion

In this study, we proposed Binarized Dual Stream VGG-16 (BDS-VGG16), which is a hardware-oriented object recognition model. We succeeded in installing it in an FPGA by reducing the memory where the weight parameters are stored, and by replacing multiplications with XNOR operations by making the weights and activation function outputs binary by referring to a BNN. The proposed BDS-VGG is inferior in terms of object recognition accuracy by only 0.6% in comparison to a conventional DS-VGG16.

We have further proposed BDS-VGG16 Tiny, a small BDS-VGG16 model, and installed it in an XZCU9EG, an SoC with a CPU, and a middle-scale FPGA on a single chip, allowing it to be installed in a service robot. As the results indicate, the processing speed was approximately 117-times higher than that of the embedded CPU, approximately 4.7-times higher than that of the CPU, and approximately 1.9-times higher than that of the GPU. It also achieved a power efficiency of approximately 114-times higher than that of the embedded CPU, approximately 20-times higher than that of the CPU, and approximately 8-times higher than that of the GPU. In addition, the proposed system installed in an ROS was proven to be capable of real-time processing.

The issues to be addressed in the future include preprocessing the depth images at a much higher speed by installing the proposed system in an FPGA, and proposing a new network capable of detecting objects as well by combining the proposed system with You Only Look Once [33] and a single-shot multibox detector [34].

Acknowledgements

This research was supported by JSPS KAKENHI Grant Numbers 19J11593 and 17H01798.

References:

[1] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara, and K. Murase, "Development of Human Support Robot as the research platform of a domestic mobile manipulator," ROBOMECH J., vol.6, no.4, 2019.

[2] Y. Ishida and H. Tamukoh, "Semi-Automatic Dataset Generation for Object Detection and Recognition and its Evaluation on Domestic Service Robots," J. Robot. Mechatron., vol.32, no.1, pp.245-253, 2020.

[3] Y. Ishida, T. Morie and H. Tamukoh, "A Hardware Intelligent Processing Accelerator for Domestic Service Robots," Advanced Robotics, vol.34, no.14, pp.947-957, June 2020.

[4] A. Magassouba, K. Sugiura, H. Kawai, "A Multimodal Target-Source Classifier with Attention Branches to Understand Ambiguous Instructions for Fetching Daily Objects," IEEE Robotics and Automation Letters, vol.5, no.2, pp.532-539, 2020.

[5] Y. Nakagawa and N. Nakagawa, "Relationship Between Human and Robot in Nonverbal Communication," J. Adv. Comput. Intell. Intell. Inform., vol.21, no.1, pp.20-24, 2017.

[6] J. Cai and T. Matsumaru, "Human Detecting and Following Mobile Robot Using a Laser Range Sensor," J. Robot. Mechatron., vol.26, no.6, pp.718-734, 2014.

[7] S. Hori, Y. Ishida, Y. Kiyama, Y. Tanaka, Y. Kuroda, M. Hisano, Y. Imamura, T. Himaki, Y. Yoshimoto, Y. Aratani, K. Hashimoto, G. Iwamoto, H. Fujita, T. Morie, and H. Tamukoh, "Hibikino-Musashi@Home 2017 Team Description Paper," arXiv:1711.05457, 2017.

[8] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "Imagenet Classification with Deep Convolutional Neural Networks," Advances in Neural Information Processing Systems, pp.1097-1105, 2012.

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, dqBackpropagation Applied to Handwritten Zip Code Recognition, Neural Computation, vol.1, no.4, pp.541-551, 1989.

[10] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Int. Conf. on Learning Representations (ICLR), 2015.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. F. Feiet, "ImageNet Large Scale Visual Recognition Challenge," International Journal of Computer Vision, vol.115, no.3, pp.211-252, 2015.

[12] A. Eitel, J.T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard, "Multimodal Deep Learning for Robust RGB-D Object Recognition," IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS) 2015, pp.681-687, 2015.

[13] Y. Yoshimoto and H. Tamukoh, "Object Recognition System using Deep Learning with Depth Images for Service Robots," International Symposium on Intelligent Signal Processing and Communication System (ISPACS) 2018, id.132, 2018.

[14] K. Lai, L. Bo, X. Ren, and D. Fox, "A Large-Scale Hierarchical Multi-View RGB-D Object Dataset," IEEE Int. Conf. on Robotics and Automation (ICRA) 2011, pp.1817-1824, 2011.

[15] H. Cheng, S. Sato and H. Nakahara, "A Performance Per Power Efficient Object Detector on an FPGA for Robot Operating System," The 9th Int. Workshop on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART), pp.1-4, 2018.

[16] Y. Tanaka and H. Tamukoh, "Hardware implementation of brain-inspired amygdala model," IEEE Int. Symposium on Circuit and Systems (ISCAS), id.2254, 2019.

[17] Y. Tanaka and H. Tamukoh, "Live Demonstration: Hardware implementation of brain-inspired amygdala model," IEEE Int. Symposium on Circuit and Systems (ISCAS), id.2351, 2019.

[18] Y. Tanaka, T. Morie, and H. Tamukoh, "An amygdala-inspired classical conditioning model on FPGA for home service robots," IEEE Access, doi: 10.1109/ACCESS.2020.3038161.

[19] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks," Advances in Neural Information Processing Systems (NIPS), vol.29, pp.4107-4115, 2016.

[20] Y. Yoshimoto and H. Tamukoh, "Live Demonstration: Hardware-Oriented Dual Stream Object Recognition System using Binarized Neural Networks," IEEE International Symposium on Circuits and Systems (ISCAS), id.2234, 2020.

[21] Y. Yoshimoto and H. Tamukoh, "Hardware-Oriented Dual Stream Object Recognition System using Binarized Neural Networks," IEEE International Symposium on Circuits and Systems (ISCAS), id.1725, 2020.

[22] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler and A. Ng, "ROS: an open-source Robot Operating System," ICRA workshop on open source software, 2009.

[23] M. Morita, T. Nishida, Y. Arita, M. Shige-eda, E. d. Maria, R. Gallone, and N. I. Giannoccaro, "Development of Robot for 3D Measurement of Forest Environment," J. Robot. Mechatron., vol.30, no.1, pp. 145-154, 2018.

[24] L. Bo, X. Ren, and D. Fox, "Unsupervised Feature Learning for RGB-D Based Object Recognition," Experimental Robotics: The 13th Int. Symposium on Experimental Robotics, pp.387-402, 2013.

[25] S. Gupta, R. Girshick, P. Arbelez, and J. Malik, "Learning Rich Features from RGB-D Images for Object Detection and Segmentation," Computer Vision – ECCV 2014, pp.345-360, 2014,

[26] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," Proc. of the 22nd ACM Int. Conf. on Multimedia, pp.675-678, ACM, 2014.

[27] H. Hagiwara, Y. Touma, K. Asami, and M. Komori, "FPGA-Based Stereo Vision System Using Gradient Feature Correspondence," J. Robot. Mechatron., vol.27, no.6, pp. 681-690, 2015.

[28] M. Muroyama, H. Hirano, C. Shao, and S. Tanaka, "Development of a Real-Time Force and Temperature Sensing System with MEMS-LSI Integrated Tactile Sensors for Next-Generation Robots," J. Robot. Mechatron., vol.32, no.2, pp. 323-332, 2020.

[29] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Proc. of the 32nd Int. Conf. on Machine Learning, vol.37, pp. 448-456, 2015.

[30] H. Yonekawa and H. Nakahara, "On-chip Memory Based Binarized Convolutional Deep Neural Network Applying Batch Normalization Free Technique on an FPGA," IEEE Int. Conf. on Field Programmable Logic and Applications (FPL) 2017, pp.1-4, 2017.

[31] H. Nakahara, T. Fujii, S. Sato "A Fully Connected Layer Elimination for a Binarized Convolutional Neural Network on an FPGA," 2017 27th International Conference on Field Programmable Logic and Applications (FPL), ISSN 1946-1488, IEEE, 2017.

[32] M. Schwarz, H. Schulz, and S. Behnke, "RGB-D Object Recognition and Pose Estimation based on Pre-trained Convolutional Neural Network Features," IEEE Int. Conf. on Robotics and Automation (ICRA) 2015, pp.13291335, IEEE, 2015.

[33] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), pp.779-788, 2016.

[34] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," European Conf. on Computer Vision, pp.21-37, 2016.

Supporting Online Materials:

[a] Xilinx Virtex UltraScale FPGA VCU110 Development Kit, https://japan.xilinx.com/products/boards-and-kits/dk-u1-vcu110-g.html, [accessed Nov. 16, 2020].

[b] Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit, https://japan.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html, [accessed Nov. 16, 2020].

[c] ROS Community Metrics Report 2020, http://download.ros.org/downloads/metrics/metrics-report-2020-07.pdf, [accessed, Dec. 1, 2020].

[d] Xilinx SDAccel Development Environment, https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug1238-sdx-rnil.pdf, [accessed Nov. 16, 2020].

[e] Intel RealSense Depth Camera D435i, https://www.intelrealsense.com/depth-camera-d435i/, [accessed Nov. 20, 2020].