PAPER
A Low-Cost Neural ODE with Depthwise Separable Convolution for Edge Domain Adaptation on FPGAs

Hiroki KAWAKAMI[†a], **Hirohisa WATANABE**[†b], **Keisuke SUGIURA**[†c], *Nonmembers,*
and **Hiroki MATSUTANI**[†d], *Member*

SUMMARY   High-performance deep neural network (DNN)-based systems are in high demand in edge environments. Due to its high computational complexity, it is challenging to deploy DNNs on edge devices with strict limitations on computational resources. In this paper, we derive a compact while highly-accurate DNN model, termed dsODENet, by combining recently-proposed parameter reduction techniques: Neural ODE (Ordinary Differential Equation) and DSC (Depthwise Separable Convolution). Neural ODE exploits a similarity between ResNet and ODE, and shares most of weight parameters among multiple layers, which greatly reduces the memory consumption. We apply dsODENet to a domain adaptation as a practical use case with image classification datasets. We also propose a resource-efficient FPGA-based design for dsODENet, where all the parameters and feature maps except for pre- and post-processing layers can be mapped onto on-chip memories. It is implemented on Xilinx ZCU104 board and evaluated in terms of domain adaptation accuracy, inference speed, FPGA resource utilization, and speedup rate compared to a software counterpart. The results demonstrate that dsODENet achieves comparable or slightly better domain adaptation accuracy compared to our baseline Neural ODE implementation, while the total parameter size without pre- and post-processing layers is reduced by 54.2% to 79.8%. Our FPGA implementation accelerates the inference speed by 23.8 times.
key words: *domain adaptation, neural ODE, distillation, FPGA, edge device*

1. Introduction

To improve the accuracy of CNNs (Convolutional Neural Networks) in image recognition tasks, a typical approach is to build deeper models by stacking more convolutional layers [1]. Although such image recognition tasks are in high demand in edge environments, computation resources are strictly limited in edge devices, making it difficult to use high-performance CNN models. To reduce the amount of parameters and mitigate this issue, light-weight neural network models have been developed [2]–[4]. Their key idea is to employ DSC (Depthwise Separable Convolution) that decomposes a conventional convolutional layer into two smaller convolutional steps.

ResNet [1] is one of conventional CNN models that stacks a lot of layers for a higher accuracy. To reduce the

parameter of ResNet, by utilizing a similarity to ODE (Ordinary Differential Equation), Neural ODE [5] repeatedly uses weight parameters instead of having a lot of different parameters. More specifically, ResNet consists of sets of layers or building blocks. An input to a building block is added to an output of the block via a shortcut connection for the residual learning. This stacking structure of ResNet is interpreted as an ODE solver, and one execution of a building block is interpreted as one step of the ODE solver. By repeatedly executing the same building block C times instead of implementing C different building blocks, the parameter size of these C blocks in ResNet is theoretically reduced to $\approx 1/C$. Thus, Neural ODE becomes significantly small compared to that of ResNet, and can be implemented in resource-limited edge devices.

Recently its implementation on a low-end FPGA (Field-Programmable Gate Array) device has been reported in [6]. However, its performance improvement is limited since only one or two building blocks are implemented on the programmable logic, and it does not employ any other parameter reduction techniques. For example, FPGA-based neural network accelerators and their optimization techniques, such as binarization and quantization, are surveyed in [7]. In [8], to fully exploit small but high-throughput on-chip memories of FPGAs, a Feature-Map-Split-CNN technique that splits a feature map into smaller patches to be stored in the on-chip memories is proposed. FPGA-optimized multipliers for DNNs that minimize information loss from quantization are studied in [9]. DSC is applied to an FPGA-based CNN accelerator in [10]. Binary neural network (BNN) is another approach to reduce memory sizes and computation costs. In [11], FracBNN that exploits fractional activations to improve the accuracy of BNNs is proposed.

In this paper, a combination of Neural ODE and DSC, called dsODENet, is proposed and implemented for FPGAs to fully utilize on-chip memory resources. As a practical use case, dsODENet is applied to domain adaptation, which is useful in a common edge AI deployment scenario. When a trained model at server side is deployed to edge devices, the distribution difference between training data and inference data acquired at edge devices often causes a performance degradation, which can be dealt with domain adaptation techniques. There are several forms of edge training scenarios, as surveyed in [12]. In this paper, we assume an edge training scenario, where the edge training is done at

edge servers (e.g., home servers and MEC servers) located between edge devices and cloud servers. The edge server has a general teacher model and datasets collected at the edge environment, leading to improved scalability and security. The edge server executes the proposed domain adaptation method so that it produces the student models optimized for the edge environment. The students models are then used in FPGA-based edge devices.

Please note that our approach is basically orthogonal to quantization techniques and can be combined with them. dsODENet is implemented on Xilinx ZCU104 board and evaluated in terms of the domain adaptation accuracy using image classification datasets, training speed, FPGA resource utilization, and speedup rate compared to a software execution [†].

The rest of this paper is organized as follows. Section 2 introduces baseline technologies behind our proposal. Section 3 introduces our domain adaptation method and Sect. 4 proposes dsODENet and describes the FPGA implementation of dsODENet. Section 5 shows evaluation results and Sect. 6 concludes this paper.

2. Related Work

2.1 Depthwise Separable Convolution

CNNs typically stack a set of convolutional layers for a higher image recognition accuracy, and each convolutional layer contains a lot of parameters. The parameter size of a convolutional layer can be calculated as a product of the number of input channels, the number of output channels, and kernel size. Let N, M, and K be the number of input channels, the number of output channels, and the length of one side of kernel, respectively. The weight parameter size of a conventional convolutional layer is NMK^2. DSC decomposes a conventional convolutional layer into two smaller convolutional steps: depthwise convolutional step and pointwise convolutional step (see Figs. 1 and 2).

In DSC, as depthwise convolutional step, as shown in Fig. 1, a convolution operation involving only spatial direction (the size is K^2) is applied for each of an input feature map. Different weight parameters are used for each of N input channels; thus its weight parameter size is NK^2. Then, an output feature map of the depthwise convolutional step (Fig. 1 right) is fed to the pointwise convolutional step as an input. As shown in Fig. 2, a 1×1 convolution operation is applied for each of the input feature map and for each of M output channels; thus its weight parameter size is NM. The weight parameter size of DSC is $NK^2 + NM$ in total, which is approximately K^2 times reduction, assuming that $N, M \gg K$.
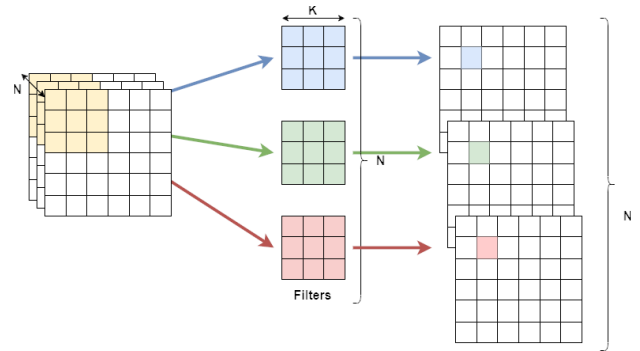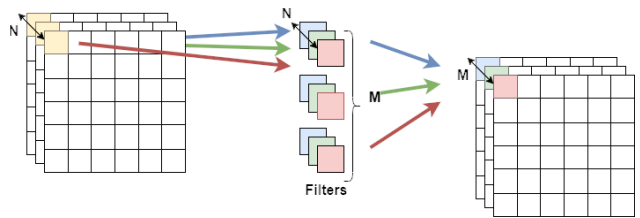

Fig. 1 Depthwise Convolutional step

Fig. 2 Pointwise Convolutional step

2.2 Ordinary Differential Equation

ODE is an ordinary differential equation that contains an unknown function and its derivatives. An example of a first-order differential equation is shown in Eq. (1).

$$\frac{dz}{dt} = f(z(t), t, \theta), \tag{1}$$

where $f(\cdot)$ is a known function and θ is the parameter. When the initial value $z(t_0)$ is given, a problem to find $z(t_1)$ that satisfies Eq. (1) is known as an initial value problem. The solution is formulated as shown in Eq. (2).

$$z(t_1) = z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta) dt \tag{2}$$

In the right side of Eq. (2), the second term contains an integral of the function, and thus it cannot be solved analytically for arbitrary functions. To solve the solution approximately, the following ODESolve function is introduced.

$$z(t_1) = ODESolve(z(t_0), t_0, t_1, f) \tag{3}$$

As an ODESolve function, Euler method shown in Eq. (4) can be used.

$$z(t_{i+1}) = z(t_i) + hf(z(t_i), t_i, \theta) \tag{4}$$

Euler method is a first-order approximation for solving the initial value problem, and it is used in this paper.

2.3 Neural ODE

ResNet is a well-known neural network architecture that can increase the number of stacked layers or building blocks by introducing shortcut connections. Using a shortcut connection, an input feature map to a building block is temporarily
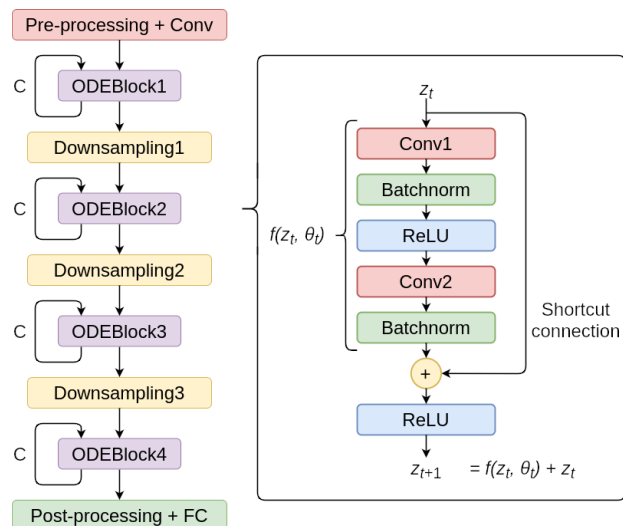
Fig. 3 ODENet architecture

saved and then it is added to the original output of the building block to generate the final output of the block. Let z_t, θ, and $f(z_t, \theta_t)$ be an input feature map to a building block, parameters of the building block, and processing result of the block, respectively. The final output of the building block is represented as Eq. (5).

$$z_{t+1} = f(z_t, \theta_t) + z_t \qquad (5)$$

Please note that Eq. (5) is similar to Eq. (4), though the former basically assumes vector values while the latter assumes scalar values. Based on this similarity, one building block can be interpreted as one step of an ODESolve function. Assuming that Euler method is used as an ODE solver, it can be interpreted that a first-order approximation is applied to solve the output of the building block. In this paper, one building block is called ODEBlock and the whole network architecture consisting of ODEBlocks is called ODENet.

Figure 3 illustrates a practical example of ODENet. The left side of Fig. 3 shows the overall ODENet architecture that consists of ODEBlocks and downsampling blocks in addition to pre-processing and post-processing layers. The pre-processing layer has a convolutional layer (denoted as Conv) and the post-processing layer has a fully-connected layer (denoted as FC). The right side of Fig. 3 shows the internal structure of an ODEBlock that consists of convolutional layers, batch normalization layers (denoted as Batchnorm), and ReLU layer. Each ODEBlock is repeatedly executed C times in ODENet, while in ResNet, C different building blocks are executed once. Let $O(L)$ be the parameter size of one building block or ODEBlock. Total parameter sizes of convolutional layers in ResNet and ODENet are $O(CL)$ and $O(L)$, respectively; thus ODENet can significantly reduce the parameter size.

2.4 Edge Domain Adaptation

Domain adaptation is a kind of transfer learning, where knowledge obtained at a source domain is transferred to a different domain called target domain. It is typically assumed that the source domain has enough labeled training data while the target domain does not. There are three domain adaptation approaches: discrepancy-based approach, adversarial-based approach, and reconstruction-based approach. The discrepancy-based approach uses fine-tuning to adapt to a target domain by reducing the domain shift. The adversarial-based approach reduces the distance between source and target distributions by using a discriminator to distinguish between source and target domains. The reconstruction-based approach assumes that reconstruction can help improving the performance of domain adaptation.

As explained in Sect. 1, it is useful in a common edge AI deployment scenario. MobileDA [14] is a discrepancy-based domain adaptation technique for edge devices based on knowledge distillation and DeepCORAL [15]. Knowledge distillation is a learning method that uses a large model and a small model. The large model is called a teacher model and the small model is called a student model. By transferring knowledge from a teacher model to a student model, the student model can acquire a similar accuracy to the teacher model. As a result, the model can be made smaller while keeping the accuracy. DeepCORAL is a method to reduce the distance between domains. In MobileDA, knowledge distillation and DeepCORAL are used when training student models, which will be detailed in the next section.

Since the target domain is an edge environment in the edge domain adaptation scenario, the target model should be further reduced in parameter size and computation cost. Although pruning, quantization, and distillation are very common model compression techniques, in this paper we propose a combination of ODENet (see Fig. 3) and DSC to further reduce the number of parameters of the target domain model.

3. Domain Adaptation Method

In this paper, a modified version of MobileDA is used as an edge domain adaptation procedure to gain a higher performance. While MobileDA uses one teacher model and one student model as knowledge distillation, our approach uses one teacher model and two student models. ResNet is used as the teacher model, while the combination of ODENet and DSC, called dsODENet, is used in the two student models. In Sect. 5, our approach will be compared to the original MobileDA.

The training phase consists of three steps. First, a teacher model is trained with source domain data in Step 1, and then two student models are trained in Steps 2 and 3. Finally, the student model makes inferences. Steps 1, 2, and 3 of the learning process are shown in Fig. 4. In Step 2, the teacher model is fixed (i.e., no update is allowed), and parameters of student model1 are trained by the help of the teacher model. In Step 3, the student model1 is fixed, and parameters of student model2 are trained by the help of the student model1. Step 3 is optional, and either student
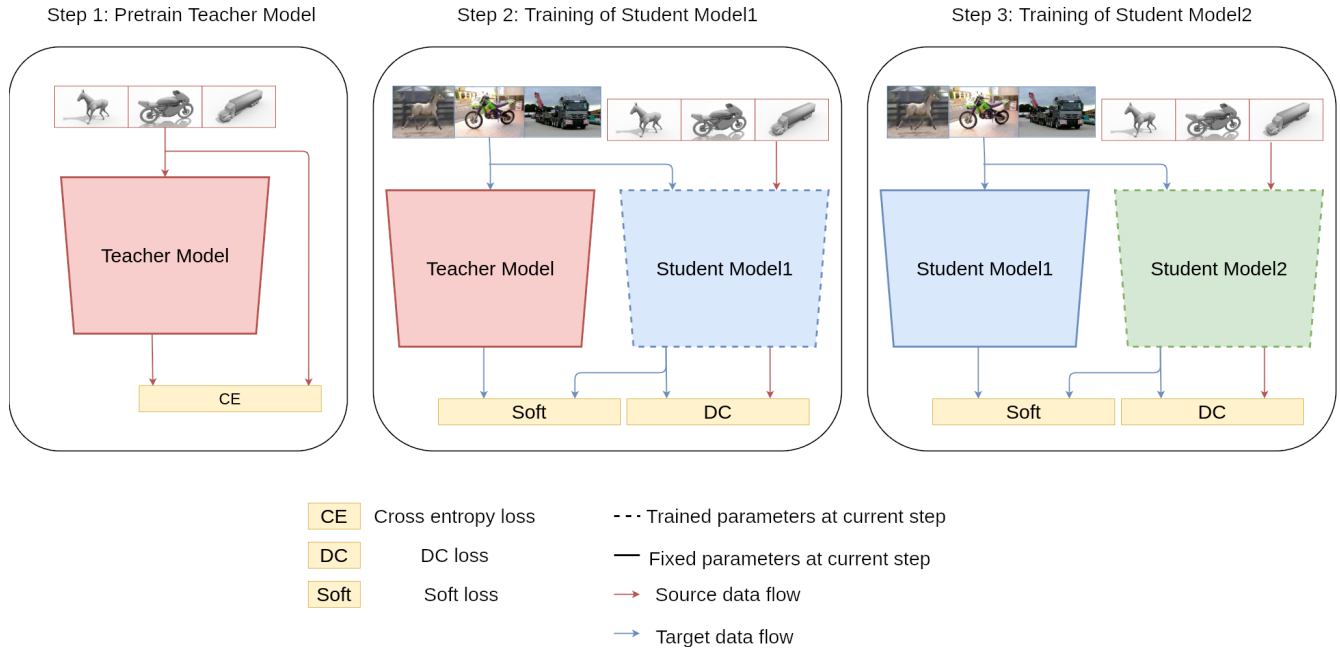
Fig. 4 Training steps of our approach. Step1: Training a teacher model on a server (left). Step2: Training a student model1 on an edge device based on the teacher model with fixed parameters (middle). Step3: Training a student model2 on an edge device based on the student model1 with fixed parameters (right).

model1 or model2 (whichever shows better accuracy) can be used for the prediction.

As shown in Fig. 4, two loss functions are used in this domain adaptation: L_{Soft} and L_{DC}. L_{Soft} is a soft target loss of knowledge distillation and it is defined in Eq. (6). L_{DC} is a loss function borrowed from DeepCORAL [15] and it is defined in Eq. (7).

$$L_{Soft} = \mathbb{E}_{x_t \sim X_t} \sum_k [L(M_T(x_t), M_S(x_t))] \quad (6)$$

$$L_{DC} = \frac{1}{4d^2} \|C_s - C_t\|_F^2, \quad (7)$$

where $L(\cdot)$ is a loss function, $M_S(x_t)$ is an output when target domain data is fed to a student model, $M_T(x_t)$ is an output when target domain data is fed to a teacher model, C_s is a covariance matrix of $M_S(x_s)$, C_t is a covariance matrix of $M_S(x_t)$, d is degree of the covariance matrix (e.g., the number of samples), and $\| \cdot \|_F^2$ is Frobenius norm, respectively. Given that target domain labels are produced by a teacher model, L_{Soft} is a loss value computed by comparing the generated target domain labels and those predicted by a student model. L_{DC} is computed by the distance between the covariance matrices of the two domains. The final loss function combines L_{Soft} and L_{DC} as shown in Eq. (8).

$$L = L_{Soft} + \lambda L_{DC} \quad (8)$$

L_{DC} is weighted by a hyper-parameter λ that controls the strength of domain confusion. A smaller λ increases the importance of class prediction results by a teacher model, which was trained by the source domain data. On the other

hand, a larger λ increases the importance of domain invariant representation, but class prediction accuracy may be weaken.

Here, target domain samples to be trained are selected by a given threshold value. Specifically, target domain samples are first fed to the teacher model in Step 2 or the student model1 in Step 3. Softmax function is then applied to the class prediction results so that the sum of the probability of each class is 1.0. If the highest class probability value of a sample is greater than a given threshold value, the sample is used for the student model training. This can prevent situations that incorrect labels produced by the teacher model in Step 2 or student model1 in Step 3 are used for the student model training. The overall flow of our domain adaptation method is summarized in Algorithm 1.

Algorithm 1 Proposed domain adaptation method

Pretrain: Training of teacher model
for each epoch **do**
  1) Obtain x_t from teacher model if the highest predicted probability value is higher than threshold
  2) Calculate the Soft Target Loss (Eq. (6))
  3) Calculate the DC Loss (Eq. (7))
  4) Train student model1 by the loss function (Eq. (8))
for each epoch **do**
  1) Obtain x_t from student model1 if the highest predicted probability value is higher than threshold
  2) Calculate the soft target loss (Eq. (6))
  3) Calculate the DC loss (Eq. (7))
  4) Train student model2 by the loss function (Eq. (8))
Output: Student model1 and student model2

---

4. dsODENet

4.1 Models

Here, we propose a resource-efficient and lightweight DNN model, termed dsODENet, that takes advantages of both ODENet and DSC for resource-limited FPGAs. Figures 5 and 6 show two dsODENet models: model with two ODE-Blocks and that with three ODEBlocks.

In Fig. 5, the right box shows an internal structure of ODEBlocks and the left box shows that of a downsampling block. The structures of ODEBlocks and downsampling blocks are similar, but in the ODEBlocks, input and output feature map sizes are the same and $M = N$, while in downsampling block, input feature map size is scaled down to $1/2 \times 1/2$ and $M = 2N$. Each ODEBlock is executed C times, while the downsampling block is executed once. In the downsampling blocks, a 1×1 convolutional operation with stride 2 (denoted as Conv) is additionally applied to the shortcut connection. To reduce the parameter size of the two ODEBlocks case, DSC can be applied to convolutional layers of ODEBlocks and downsampling block. We observed that the accuracy is sometimes sensitive to the DSC on the downsampling block that rescales the feature map. Considering the stability, in the two ODEBlocks case, DSC is applied to the two ODEBlocks while it is not applied to the downsampling block, as shown in Fig. 5. Assuming that N is 64 in ODEBlock1, total parameter size of ODEBlock1, downsampling1, and ODEBlock2 in the ODENet without DSC is 598,016. In dsODENet, the total size is 273,792, which is 54.2% reduction. Assuming a 32-bit fixed-point representation, their sizes are 19.1 Mb and 8.8 Mb, respectively.

As shown in Fig. 6, the three ODEBlocks case has three ODEBlocks and two downsampling blocks. DSC is applied to all the three ODEBlocks. Regarding the downsampling blocks, their parameter sizes without DSC are 221,184 and 884,736, respectively, assuming that N is 64 in ODEBlock1. Their sizes depend on the numbers of input and output channels, and these numbers are doubled once the downsampling is applied; thus, the second downsampling block (Downsampling2) is much larger than that of the first one (Downsampling1). Since DSC on the downsampling block is sometimes sensitive to accuracy as mentioned above, in this paper DSC is applied only to the second downsampling block. Please note that the total parameter size of ODEBlock1, Downsampling1, ODEBlock2, Downsampling2, and ODEBlock3 in the ODENet without DSC is 2,695,168. In dsODENet, the total parameter size is 544,000, which is 79.8% reduction. When a 32-bit fixed-point representation is employed, their sizes are 86.2 Mb and 17.4 Mb, respectively. This parameter size reduction by DSC is significant since these parameters can be implemented on BRAM or URAM of modest FPGA devices for simplicity.

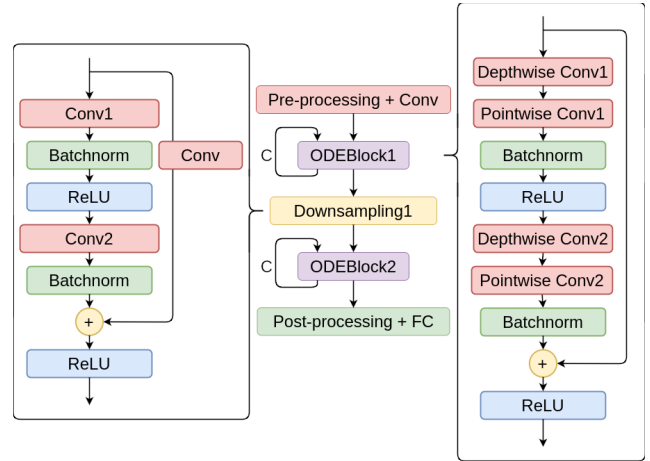In Sect. 5, ResNet, ODENet, and dsODENet are used
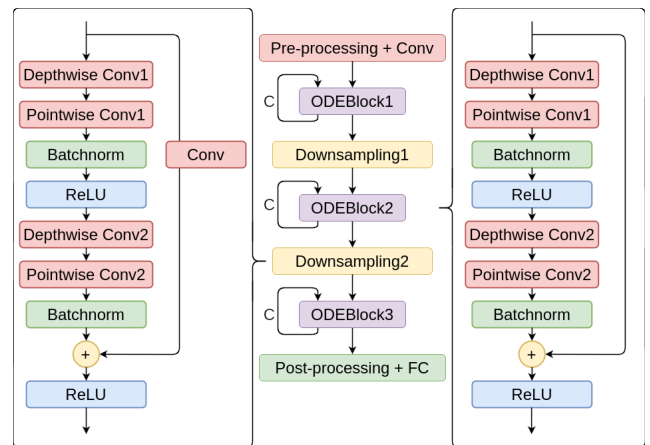

Fig. 5 Model with two ODEBlocks

Fig. 6 Model with three ODEBlocks

as student models of the domain adaptation as mentioned in Sect. 3. In this paper, ResNet-50 [1] is used as a baseline. C is set to 10 so that the number of convolutional layers of ODENet/dsODENet is same as that of ResNet-50.

4.2 FPGA Accelerator

This section describes the design of a dsODENet accelerator using FPGA SoC. As mentioned in Sect. 3, dsODENet repeatedly executes ODEBlocks, which is a bottleneck in the execution time. Thus, all the ODEBlocks and intermediate processing between them are accelerated by the programmable logic of the FPGA.

4.2.1 Top Module

Figure 7 illustrates a block diagram of the board-level implementation, which is divided into a processing system (PS) part and a programmable logic (PL) part. The proposed dsODENet IP core and a direct memory access (DMA) controller are instantiated in the PL part. The PS part sets up the IP core and transfers input images to the PL part by
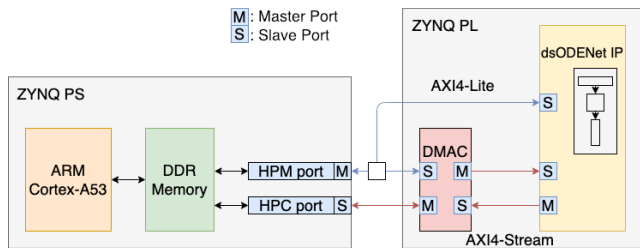
Fig. 7 Board-level implementation

the DMA controller, while the PL part computes feature maps of the incoming images sent from the PS part. The PS part is also in charge of pre-processing and post-processing parts of dsODENet. For high-speed data transfer, the DMA controller is connected to a 32-bit wide high-performance slave port (HPC port) in the AXI4-Stream protocol (red line in Fig. 7). The control registers are connected to a high-performance master port (HPM port) via the AXI4-Lite interface (blue line in Fig. 7).

The dsODENet core has two modes: weight transfer mode and feature map computation mode. In the weight transfer mode, the IP core receives weight parameters of dsODENet via the AXI4-Stream interface and stores them in on-chip BRAM and URAM buffers. The transfer mode is finished when the IP core sends back a 32-bit nonzero value as an acknowledge message. In the feature map computation mode, a pre-processed input image ($\mathbb{R}^{8 \times 8 \times 3}$) is sent to the IP core, and then it is processed by ODEBlock1, Downsampling1, ODEBlock2, Downsampling2, and ODEBlock3 in sequence to produce the final feature map ($\mathbb{R}^{1 \times 1 \times 256}$).

These ODEBlocks are iteratively executed by feeding back the output of an iteration step to the input of the next step. Since the weight parameters used in the iterations are the same, utilizations of internal memories (BRAM and URAM) can be significantly reduced. In addition, the design is scalable in terms of the number of iterations, since the hardware resources are constant even if the number of iterations is increased. Please note that the weight parameters and feature maps are packed in the internal memories by quantizing them with 24-bit or 20-bit fixed-point formats, as analyzed in Sect. 5.3.1. By the quantization, these blocks are running efficiently without accessing on-board DRAM modules.

4.2.2 dsODENet and ODEBlock Modules

As shown in Fig. 8, the dsODENet IP core consists of five major modules: ODEBlock1, Downsampling1, ODEBlock2, Downsampling2, and ODEBlock3. The number of output channels is increased while the output feature map size (width and height) is decreased at each Downsampling block. That is, output feature map sizes of ODEBlock1, Downsampling1, ODEBlock2, Downsampling2, and ODEBlock3 are $8 \times 8 \times 64$, $4 \times 4 \times 128$, $4 \times 4 \times 128$, $2 \times 2 \times 256$, $2 \times 2 \times 256$, respectively. As shown in Fig. 8, each ODEBlock consists of two sets of AddTime, DepthwiseConv,
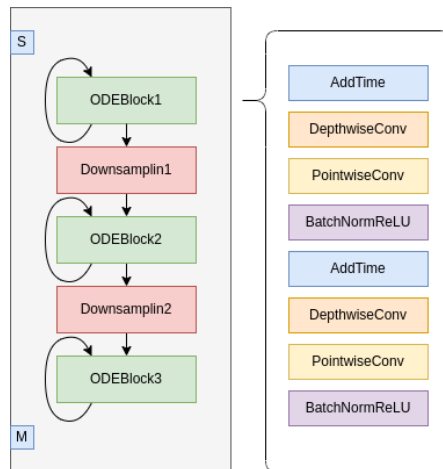

Fig. 8 computation of dsODENet IP core

PointwiseConv, BatchNorm, and ReLU modules. AddTime module inserts a new channel representing the current iteration count to the feature map. That is, it increases the number of channels by one, and its feature map buffer is also extended to store one more channel. Since ODEBlocks have a skip-connection structure, three feature map buffers are implemented in each ODEBlock: two for input and output buffers of the convolutional modules and one for saving the original input feature map.

Loop unrolling and loop pipelining directives are used in DepthwiseConv, PointwiseConv, ReLU, and BatchNorm modules. The unrolling factor is set to eight in these modules, so they are processed by eight arithmetic units in parallel.

4.2.3 Implementation

dsODENet IP core was designed and implemented with Xilinx Vitis HLS 2020.2 and Xilinx Vivado 2020.2 for the high-level synthesis and place-and-route of the design. We chose Xilinx Zynq UltraScale+ MPSoC series as a target FPGA platform, and ZCU104 evaluation board kit (XCZU7EV-2FFVC1156) was used in this paper. The CPU on the FPGA device is ARM Cortex-A53 running at 1.2 GHz. We used Python 3.8.2 and PyTorch 1.10.2 in the software stack.

The weight parameters of dsODENet are trained by a domain adaptation method described in Sect. 3. The trained parameters are then transferred to the DRAM of the target FPGA SoC. To start an inference processing at the PL part, the parameters are moved to internal BRAM and URAM buffers in the weight transfer mode. In the feature map computation mode, an input feature map is transferred to the PL part, it is processed at the PL part, and then the output feature map is sent back to the PS part. Please note that the parameters and feature maps are implemented on the BRAM and URAM of the FPGA to fully enjoy benefits of using fast on-chip memories. A larger batch size is typically used in the training phase for a fast training. In contrast, this inference accelerator assumes a single batch size since the

inference processing is triggered by a per-request basis.

Although the BRAM and URAM sizes are 11 Mb and 27 Mb in total, respectively, their instance sizes are 36 kb and 288 kb, which means that, depending on the number and sizes of parameter arrays, a part of BRAM and URAM instances is underutilized. In our design, each parameter array is carefully implemented on either BRAM or URAM instances to minimize the underutilized on-chip memories. More specifically, in the three ODEBlocks case, parameter arrays of normal convolutional layers of Downsampling1, those of depthwise and pointwise convolutional layers of Downsampling2, and those of pointwise convolutional layers of ODEBlock3 are implemented on the URAM instances; and the others are implemented on BRAM instances.

5. Evaluations

The proposed dsODENet for FPGAs is evaluated with an edge domain adaptation scenario using image recognition datasets. For accuracy evaluations, it is implemented with Pytorch 1.8.1 and torchvision 0.9.1. For resource utilization and performance evaluations, it is implemented with Xilinx Vivado v2020.2 for ZCU104 FPGA board. Specification of the board is shown in Table 1.

5.1 Number of Parameters

A comparison of the number of parameters for ResNet, ODENet, and dsODENet is shown in Tables 2 and 3. The percentage compared to the number of parameters in ResNet is listed in ODENet and dsODENet. The reduction in the number of parameters from the third building block, which has a large number of parameters in ResNet, to ODEBlock3 in dsODENet is significant. In total, dsODENet reduces the

number of parameters of these blocks by 96.7% compared to ResNet.

A comparison of the number of parameters in the two building-block case (denoted as Block2) and the three building-block case (denoted as Block3) for ResNet, ODENet, and dsODENet is shown in Fig. 9 in a logarithmic scale. As shown, the number of parameters is greatly reduced in the order of ResNet, ODENet, and dsODENet. Also, the three ODEBlocks case in dsODENet has fewer parameters than the two ODEBlocks case in ODENet.

5.2 Accuracy

For accuracy evaluations, Office-31 dataset (Office-31 [16]), road signs datasets (Synth signs [17] and GTSRB [18]), and digit datasets (SVHN [19] and MNIST [20]) are used as the datasets. Their input image sizes are 256×256, 40×40, and 32×32, respectively. The numbers of their classes are 31, 43, and 10. The amounts of their domain shifts are various, small, and large. The MNIST images are grayscale, while SVHN images are RGB colored; thus the MNIST images are duplicated for three channels so that they are compatible with the 3-channel SVHN images. Office-31 is a popular dataset used for domain adaption tasks. It contains 4,110 images, and they are divided into three domains: Amazon (A-domain), Webcam (W-domain), and DSLR (D-domain). The numbers of their images are 2817, 795, and 498, respec-

Table 3 Numbers of parameters of ODENet and dsODENet

		ODENet		dsODENet	
ODEBlock1	Conv1	36,864	(10.0%)	4,672	(1.3%)
	Conv2	36,864	(10.0%)	4,672	(1.3%)
Downsampling1	Conv	8,192	(100.0%)	8,192	(100.0%)
	Conv1	73,728	(100.0%)	73,728	(100.0%)
	Conv2	147,456	(100.0%)	147,456	(100.0%)
ODEBlock2	Conv1	147,456	(10.0%)	17,536	(1.2%)
	Conv2	147,456	(10.0%)	17,536	(1.2%)
Downsampling2	Conv	32,768	(100.0%)	32,768	(100.0%)
	Conv1	294,912	(100.0%)	33,920	(11.5%)
	Conv2	589,824	(100.0%)	67,840	(11.5%)
ODEBlock3	Conv1	589,824	(10.0%)	67,840	(1.2%)
	Conv2	589,824	(10.0%)	67,840	(1.2%)
Others		1,664		1,664	
Total		2,696,832	(16.2%)	545,664	(3.3%)

Table 1 Specification of ZCU104 board

OS	Pynq Linux (Ubuntu 18.04)
CPU	ARM Cortex-A53 @ 1.2 GHz
DRAM	DDR4 2 GB
FPGA	Zynq UltraScale+ XCZU7EV-2FFVC1156

Table 2 Number of parameters of ResNet

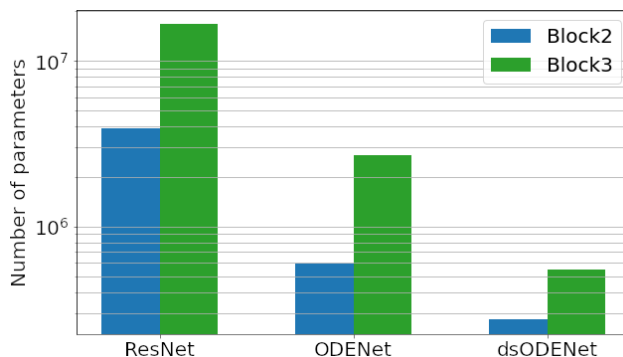|		ResNet
Building block1	Conv1	368,640
	Conv2	368,640
Downsampling1	Conv	8,192
	Conv1	73,728
	Conv2	147,456
Building block2	Conv1	1,474,560
	Conv2	1,474,560
Downsampling2	Conv	32,768
	Conv1	294,912
	Conv2	589,824
Building block3	Conv1	5,898,240
	Conv2	5,898,240
Others		9,728
Total		16,639,488

Fig. 9 Comparison of the numbers of parameters of three models

Table 4 Number of parameters of models

Mode	Number of parameters
dsODENet	595,072
AlexNet	61,100,840
CDTrans [22]	86,578,408
DSN [23]	1,102,159
DFA [24]	31,493,150
ADDA [25]	426,070
M-T [26]	4,338,283

Table 5 Domain adaptation accuracy of Office-31 dataset

Model	A→W (%)	A→D (%)
CDAN [21]	77.9	75.1
MobileDA [14]	71.5	75.3
CDTrans [22]	97.0	96.7
Teacher model ResNet-50	75.8	78.3
Student model1 ResNet-50	80.6	80.9
Student model1 ODENet	71.3	78.8
Student model1 dsODENet	80.4	79.1
Student model2 dsODENet	83.2	79.1

tively. A→W means a domain adaptation scenario in which A-domain is a source domain and W-domain is a target domain. The numbers of W-domain and D-domain images are smaller than that of A-domain. A→W and A→D scenarios are examined in this paper because domain adaptation from a domain with more images to that with less images is a typical use case. The edge domain adaptation procedure proposed in Sect. 3 is used. All the labeled source domain data and all the unlabeled target domain data are used for the training phase. The accuracy is then evaluated with all the labeled target data.

Either SGD or Adam (whichever shows better accuracy) is selected as an optimizer. The learning rate is reduced based on Eq. (9).

$$\eta = \frac{\eta_0}{(1 + \alpha p)^\beta}, \tag{9}$$

where $\eta_0 = 0.01$, $\alpha = 10$, $\beta = 0.75$, and p is linearly changed from 0 to 1. In the evaluations, the same experiments are executed three times and their accuracy values are averaged and reported. As for a teacher model, an initial model was pre-trained with ImageNet dataset, and using this initial model without the final fully-connected layer, the teacher model is trained. The learning rate is reduced to 1/10 for these pre-trained layers. Different student models that use ResNet-50, ODENet, and dsODENet are compared in terms of accuracy. ODENet and dsODENet use three ODE-Blocks as shown in Fig. 6. The number of executions C of an ODEBlock is set to 10. They are also compared to a teacher model of ResNet-50 and other domain adaptation techniques [14], [21]–[26].

Table 4 compares the parameter sizes of dsODENet, AlexNet, CDTrans, DSN, DFA, ADDA, and M-T. Please note that the accuracies of CDTrans, DFA, and M-T are higher than the proposed approach as shown in Tables 5–7, only the dsODENet and ADDA models could be implemented solely using BRAM/URAM slices of the target FPGA board (Xilinx ZCU104 board) due to the their scarcity. Please note tha, CDTranst DSN, DFA, ADDA, and M-T are domain adaptation methods but they do not rely on the knowledge distillation; thus they do not use student models.

5.2.1 Office-31 Dataset

As a counterpart, MobileDA uses AlexNet as a student model and ResNet-50 as a teacher model. 80% of target domain data is used for the training. As another counterpart, CDAN [21] is also considered, which is a domain adaptation technique based on adversarial training. In addition, CDTrans [22] is a domain adaptation method using a transformer. Table 5 shows the evaluation results of CDAN, MobileDA, and our approach with different student models.

"Student model1 dsODENet" and "Student model2 dsODENet" are our proposed models, where dsODENet is used for student model1 and student model2, respectively. As shown in the table, the accuracy of A→W is improved by 4.6% and 2.8% for the teacher model to "Student model1 dsODENet" and "Student model1 dsODENet" to "Student model2 dsODENet", respectively. Also, the accuracy of A→D is improved by 0.8% for the teacher model to "Student model1 dsODENet". However, there is no improvement in accuracy from the student model1 to the student model2.

Please note that the proposed approach differs from the original MobileDA in the following points. The proposed approach does not use hard target loss in the loss function because in our evaluation environment the accuracy was slightly improved by not using it. In the proposed approach, two student models are used. That is, a student model1 is trained from a teacher model, and then a student model2 is trained from the student model1. In this paper, we assume that "Student model1 dsODENet" corresponds to the results of MobileDA when dsODENet is used as a student model. In other words, accuracy differences between "Student model1 dsODENet" and "Student model2 dsODENet" are improvements obtained by the proposed approach. As shown in Tables 5, 6, and 7, the proposed approach actually improves the accuracy in most cases. Accuracy of CDTrans outperforms the proposed approach while its parameter size is much larger than the proposed one.

Figure 10 shows the training speeds of different student models (ResNet-50, ODENet, and dsODENet) for A→D scenario. As shown in the figure, the student model of ResNet-50 is converged faster than the others, followed by those of dsODENet and ODENet. In ResNet-50, the number of implemented building blocks is optimized for each feature map size in this experiment. These numbers in ResNet-50 are interpreted as the numbers of continuous executions of ODEBlocks in the cases of ODENet and dsODENet (C in Figs. 5 and 6). As mentioned, in ODENet and dsODENet, a single execution of an ODEBlock is interpreted as a single step of Euler method. Reducing the number of executions C

Table 6 Domain adaptation accuracy of road sign datasets

Model	Synth signs→GTSRB (%)
DSN [23]	93.1
DFA [24]	97.5
Teacher model ResNet-50	95.1
Student model1 ResNet-50	97.1
Student model1 ODENet	97.0
Student model1 dsODENet	97.1
Student model2 dsODENet	97.3

Table 7 Domain adaptation accuracy of digit datasets

Model	SVHN→MNIST (%)
ADDA [25]	76.0
M-T [26]	93.3
Teacher model ResNet-50	76.5
Student model1 ResNet-50	82.6
Student model1 ODENet	82.5
Student model1 dsODENet	83.5
Student model2 dsODENet	86.6
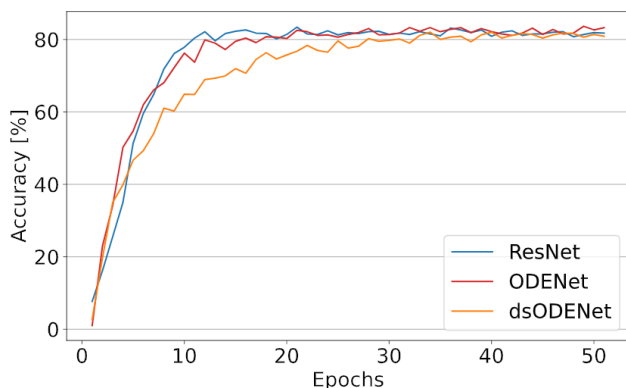
Fig. 10 Training speed of different student models in Offce-31 dataset

of an ODEBlock degrades the approximation performance and reduces the accuracy. To obtain a stable accuracy, the number of continuous executions C of an ODEBlock is equally set to 10 for each feature map size in ODENet and dsODENet cases, though there is still a room for optimization. Please note that the training speed of dsODENet is faster than ODENet, because in dsODENet the number of parameters to be trained is reduced by 54.2% to 79.8% by DSC.

5.2.2 Road Sign Dataset

As a counterpart, DSN [23] which is a domain adaptation based on adversarial training is compared to our approach with different student models. In addition, DFA [24] employs a latent alignment approach based on the encoder-decoder formulation. Table 6 shows the evaluation results of DSN and our approach with different student models. Please note that network structures of DSN and ours are different and a fair comparison is difficult. Accuracy of DFA is higher than that of the proposed approach and DSN while its parameter size is larger than the proposed one.

"Student model1 dsODENet" and "Student model2

dsODENet" are our proposed models, where dsODENet is used for student model1 and student model2, respectively. As shown in the table, the accuracy of Synth signs→GTSRB is improved by 2.0% and 0.2% for the teacher model to "Student model1 dsODENet" and "Student model1 dsODENet" to "Student model2 dsODENet", respectively. The accuracy of "Student model1 dsODENet" becomes higher than that of the teacher model because the two datasets contain a large enough number of images and their domain shift is small. In contrast, the improvement in accuracy from "Student model1 dsODENet" to "Student model2 dsODENet" is small. As the domain shift is small, the accuracies of ResNet-50, ODENet, and dsODENet are similar.

5.2.3 Digit Dataset

As a counterpart, ADDA [25] which is also a domain adaptation based on adversarial training is considered. In addition, Mean teacher (M-T) [26] is an approach using a self-ensembling technique.

Table 7 shows the evaluation results of ADDA and our approach with different student models. As shown in the table, there is an improvement of 7.0% and 1.7% in accuracy from teacher model to "Student model1 dsODENet" and from "Student model1 dsODENet" to "Student model2 dsODENet". Accuracy of M-T is higher than that of the proposed approach and ADDA while its parameter size is larger than the proposed one. We consider that this two-step improvement in accuracy is the result of both the domain adaptation and knowledge distillation. As shown in Algorithm 1, target domain samples to be used for knowledge distillation are selected by a given threshold value. If the threshold value is low, the number of training samples for a student model1/2 increases, while it becomes more likely that samples mislabeled by a teacher model/student model1 are used during training. We set a higher threshold value in the first knowledge distillation from a teacher model to a student model1. We observed a lower threshold causes a training instability because this is an adaptation from a larger model to a smaller model. We set a lower threshold value in the second knowledge distillation from the student model1 to a student model2. Since their model sizes are the same, we observed that the lower threshold can increase the number of training samples and produce slightly better accuracy, as reported in this section. Please note that adding the second-step is meaningful while adding the third-step does not have a similar impact if the threshold value is the same.

Figure 11 shows the training speeds of different student models (ResNet-50 and dsODENet) for SVHN→MNIST scenario. In Fig. 11, six lines are shown in total since ResNet-50 and dsODENet are trained three times each. dsODENet is converged fast and stable compared to ResNet-50 because the number of parameters of dsODENet is significantly reduced compared to ResNet-50. For domain adaptation tasks with a larger domain shift, the accuracy of ResNet-50 becomes higher than that of dsODENet but the
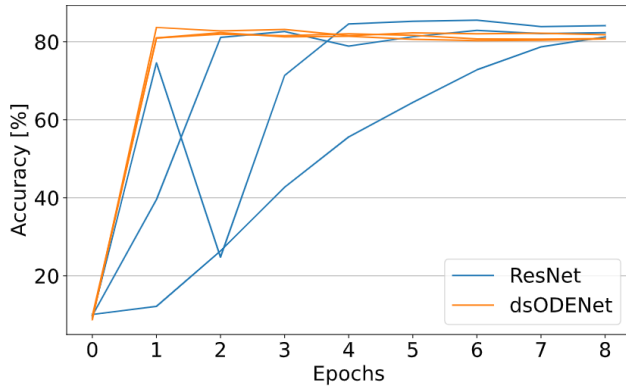
Fig. 11   Training speed of different student models in digit datasets

Table 8   Domain adaptation accuracy of digit datasets with quantization

Model	SVHN→MNIST (%)
Teacher model ResNet-50	76.51
Student model2 dsODENet w/o quantization	86.57
Student model2 dsODENet w/ quantization	86.55

training curve of dsODENet is still stable.

5.3   FPGA Evalutation

We develop a dsODENet accelerator by offloading the computation onto the FPGA SoC. Since a quantization is applied in our FPGA implementation, in this section it is evaluated in terms of the inference accuracy, resource utilization, and performance on the FPGA.

5.3.1   Effect of Quantization

In the FPGA implementation, a quantization is applied to pack the weight parameters and feature maps in the BRAM and URAM slices. More specifically, a 20-bit fixed-point format is used in the convolutional layers, while a 24-bit fixed-point format is used in the other layers. It is thus expected that the actual accuracy of the FPGA implementation slightly differs from those evaluated in Sect. 5.2. Table 8 shows evaluation results of the accuracy with digit Dataset when the quantization is applied. We evaluated the effect of quantization only with the digit datasets. This is because only the dsODENet models for relatively small inputs (digit images) could be implemented solely using BRAM/URAM slices due to their scarcity. Because the other datasets (e.g., Office-31) contain larger images, which necessitates more aggressive quantizations and manual optimizations in order to implement dsODENet models for those larger inputs on the target FPGA. Since BRAM/URAM sizes of FPGA devices are continually increasing, such optimized implementations for larger datasets are our future work. The results show that the accuracy drops by 0.02% but this accuracy loss is much smaller than the accuracy improvement obtained by the proposed domain adaptation method compared to the original teacher model.

Table 9   FPGA resource utilization of dsODENet

	BRAM	DSP	FF	LUT	URAM
Block2	208 (67%)	760 (44%)	22,880 (5%)	63,461 (28%)	64 (67%)
Block3	265 (85%)	138 (8%)	19,755 (4%)	49,014 (21%)	92 (96%)

Table 10   Execution time of each block on FPGA

	FPGA (ms)	CPU (ms)	Speedup
ODEBlock1	7.30	112.85	15.5
Downsampling1	5.14	16.13	3.1
ODEBlock2	6.42	213.42	33.2
Downsampling2	1.46	34.56	23.7
ODEBlock3	4.84	396.98	82.0
DMA transfer (PL)	0.13	-	-
DMA transfer (PS)	7.24	-	-
Total	32.53 ± 1.58	773.94 ± 35.0	23.8

5.3.2   Resource Utilization

Table 8 shows FPGA resource utilizations when two ODE-Blocks case (Block2) and three ODEBlocks case (Block3) are implemented. Importantly, weight parameter and feature map arrays of the Block3 case can be implemented on the BRAM and URAM slices of the FPGA without using external DRAMs. The results also show that the Block2 implementation still has room in the BRAM and URAM capacities, while they are close to 100% in the Block3 implementation, which means that there is room for improving the performance and accuracy in the Block2 implementation by introducing more aggressive parallelization and wider bit widths.

5.3.3   Execution Time

Table 10 compares the execution time of each block for a single data sample between our FPGA implementation and its software counterpart (denoted as FPGA and CPU). The mean and standard deviation of the execution time to infer one image is calculated by inferring 100 images 30 times. The execution time for each block in the FPGA is calculated from the number of cycles of the high-level synthesis result. In the FPGA case, a DMA transfer between PS–PL is additionally required, while the high-level synthesis result takes into account only for the PL part. Thus, there is an execution time difference between the high-level synthesis result and the actual on-board execution time that takes into account the DMA transfer in the PS part, which is denoted as DMA transfer (PS) in Table 10. Both the ODEBlocks and downsampling blocks were accelerated by 3.1-82.0 times, which contributes to the overall speedup of 23.8 times and 30.6 times with and without considering the DMA transfer (PS). If we focus on a single execution time of each block, the downsampling blocks achieved better speedups than ODEBlocks, as they involve normal convolution, which is more compute-intensive than DSC. However, ODEBlocks are repeatedly executed C times and thus their impact is higher than those of Downsampling1 and Downsampling2, as shown in Table 10. Further performance im-

provement of the FPGA can be expected by more aggressive parallelization of multiply-add operations.

Please note that an advantage of our approach is that dsODENet is a ResNet-like backbone architecture which can be stored in small but high-throughput on-chip memories of FPGAs. It is thus orthogonal to other promising techniques, such as split-CNN technique [8] and aggressive quantization including BNNs [11]. Another advantage of our approach is that, by increasing C parameter of ODENet/dsODENet, we can increase the number of iterations of ODEBlocks/dsODEBlocks to improve the accuracy without increasing the number of parameters.

6. Conclusion

In this paper, a combination of Neural ODE and DSC, called dsODENet, is proposed and implemented for FPGAs. dsODENet is applied to a distillation-based edge domain adaptation as student models. All the dsODENet blocks except the pre- and post-processing layers are implemented on PL part of Xilinx ZCU104 FPGA board and the others are executed on PS part. Importantly, all their parameter and feature map arrays are stored in URAM and BRAM instances of the FPGA without relying on external DRAMs. It is evaluated in terms of domain adaptation accuracy, training speed, FPGA resource utilization, and speedup rate compared to a software execution. Regarding the domain adaptation accuracy, the student models of dsODENet are comparable to or better than that of ODENet and better than some existing domain adaptation methods. The total parameter size of dsODENets without pre- and post-processing layers is reduced by 54.2% to 79.8%. The FPGA implementation accelerates the prediction tasks by 23.8 times than a software implementation running on PS part.

As a future work, we are optimizing our FPGA implementations by introducing 16-bit fixed-point or bfloat16 representations to suppress the on-chip memory utilizations and enable more aggressive parallel implementation.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.770–778, 2016.

[2] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.1800–1807, July 2017.

[3] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," arXiv:1704.04861, 2017.

[4] A. Howard, M. Sandler, B. Chen, W. Wang, L.C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q.V. Le, "Searching for MobileNetV3," Proc. International Conference on Computer Vision (ICCV'19), pp.1314–1324, Oct. 2019.

[5] R.T.Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," Proc. Annual Conference on Neural Information Processing Systems (NeuroIPS'18), pp.6572–6583, Dec. 2018.

[6] H. Watanabe and H. Matsutani, "Accelerating ODE-based neural networks on low-cost FPGAs," Proc. IEEE International Parallel and Distributed Processing Symposium (IPDPS'21) Workshops, pp.88–95, March 2021.

[7] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of FPGA-based neural network accelerator," arXiv:1712.08934v3, Dec. 2018.

[8] A. Jinguji, S. Sato, and H. Nakahara, "Weight sparseness for a feature-map-split-cnn toward low-cost embedded fpgas," IEICE Trans. Inf. & Syst., vol.E104-D, no.12, pp.2040–2047, Dec. 2021.

[9] J. Faraone, M. Kumm, M. Hardieck, P. Zipf, X. Liu, D. Boland, and P.H. Leong, "AddNet: Deep neural networks using FPGA-optimized multipliers," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.28, no.1, pp.115–128, Jan. 2020.

[10] L. Bai, Y. Zhao, and X. Huang, "A CNN accelerator on FPGA using depthwise separable convolution," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol.65, no.10, pp.1415–1419, Oct. 2018.

[11] Y. Zhang, J. Pan, X. Liu, H. Chen, D. Chen, and Z. Zhang, "FracBNN: Accurate and FPGA-efficient binary neural networks with fractional activations," Proc. International Symposium on Field Programmable Gate Arrays, (FPGA21), pp.171–182, 2021.

[12] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," Proc. IEEE, vol.107, no.8, pp.1738–1762, Aug. 2019.

[13] H. Kawakami, H. Watanabe, K. Sugiura, and H. Matsutani, "dsODENet: Neural ODE and depthwise separable convolution for domain adaptation on FPGAs," Proc. Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP'22), pp.152–156, March 2022.

[14] J. Yang, H. Zou, S. Cao, Z. Chen, and L. Xie, "MobileDA: Toward edge-domain adaptation," IEEE Internet Things J., vol.7, no.8, pp.6909–6918, Aug. 2020.

[15] B. Sun and K. Saenko, "Deep CORAL: Correlation alignment for deep domain adaptation," arXiv:1607.01719, 2016.

[16] K. Saenko, B. Kulis, M. Fritz, and T. Darrell, "Adapting visual category models to new domains," Proc. European Conference on Computer Vision (ECCV'10), vol.6314, pp.213–226, Sept. 2010.

[17] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," Proc. International Conference on Machine Learning (ICML'15), pp.1180–1189, July 2015.

[18] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The German traffic sign recognition benchmark: A multi-class classification competition," Proc. International Joint Conference on Neural Networks (IJCNN'11), pp.1453–1460, Aug. 2011.

[19] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng, "Reading digits in natural images with unsupervised feature learning," Proc. NIPS Workshop on Deep Learning and Unsupervised Feature Learning, Dec. 2011.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proc. IEEE, vol.86, no.11, pp.2278–2324, Nov. 1998.

[21] M. Long, Z. Cao, J. Wang, and M.I. Jordan, "Conditional adversarial domain adaptation," Proc. Annual Conference on Neural Information Processing Systems (NeuroIPS'18), pp.1640–1650, Dec. 2018.

[22] T. Xu, W. Chen, P. Wang, F. Wang, H. Li, and R. Jin, "CD-Trans: Cross-domain transformer for unsupervised domain adaptation," Proc. 10th International Conference on Learning Representations, (ICLR'22), April 2022.

[23] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan, "Domain separation networks," Proc. Annual Conference on Neural Information Processing Systems (NeuroIPS'16), pp.343–351, Dec. 2016.

[24] J. Wang, J. Chen, J. Lin, L. Sigal, and C.W. de Silva, "Discriminative feature alignment: Improving transferability of unsupervised domain adaptation by gaussian-guided latent alignment," Pattern Recognit., vol.116, Article No. 107943, Aug. 2021.

[25] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," Proc. IEEE Conference on Com-

puter Vision and Pattern Recognition (CVPR'17), pp.2962–2971, July 2017.

[26] G. French, M. Mackiewicz, and M. Fisher, "Self-ensembling for visual domain adaptation," Proc. International Conference on Learning Representations (ICLR'18), 2018.

Hiroki Kawakami received the B.E. and M.E. degree from Keio University in 2021 and 2023 respectively.

Hirohisa Watanabe received the B.E. and M.E. degree from Keio University in 2020 and 2022 respectively.

Keisuke Sugiura received the B.E. and M.E. degree from Keio University in 2020 and 2022 respectively. He is currently a doctor course student in Keio University.

Hiroki Matsutani received the B.A., M.E., and Ph.D. degrees from Keio University in 2004, 2006, and 2008, respectively. He is currently a professor in the Department of Information and Computer Science, Keio University. His research interests include the areas of computer architecture and interconnection networks.