FPGA Implementation of 3-bit Quantized CNN for Semantic Segmentation

MIYAMA, Masayuki
Kanazawa University, Kakumamachi, Kanazawa, Ishikawa, 920-1192, Japan

e-mail: miyama@se.kanazawa-u.ac.jp

Abstract. Semantic segmentation is a task of inputting an image and performing category classification for each pixel. Semantic segmentation by CNN has high accuracy but its calculation using floating-point numbers consumes a large amount of power. We adopted UNET as the semantic segmentation CNN and improved it for FPGA implementation. We quantized both weights and activations of the network up to 3-bit. Then, we devised a dedicated hardware architecture for the quantized CNN and implemented it on an FPGA. This circuit uses only internal memory to perform forward propagation calculations, that eliminates high-power external memory accesses. This circuit is a stall-free pixel-by-pixel pipeline, and performs 8 rows, 16 input channels, 16 output channels, 3 by 3 pixels convolution calculations in parallel. The convolution calculation performance at an operating frequency of 300 MHz is 11 TOPs/s.

1. Introduction

Recently, AI technology using DNN (Deep Neural Network) is rapidly developing. Category classification is a task to determine the type of object shown in the image. Semantic segmentation is the category classification for each pixel, and outputs an image in which a label indicating a category is assigned to each pixel. By using CNN (Convolutional Neural Network) which is a kind of DNN, these tasks can be executed with high accuracy.

The amount of computation of DNN is enormous, and a large amount of power is required for learning and inference. DNN calculations are usually performed with floating-point numbers, but low bit quantization and dedicated hardware are being developed to reduce power consumption [1,2]. If we quantize up to 1 bit, the product of convolution can be done with XNOR and the sum can be done with popcount (counting 1), so we can realize hardware with high power efficiency [3,4]. Most of the low bit quantization of DNN and its hardware implementation have been aimed at category classification, but researches on semantic segmentation have been also done [5,6,7,8,9].

This research adopted UNET and modified it for FPGA implementation. UNET is a CNN for semantic segmentation, which features a skip connection that connects the same layers of encoder and decoder [10]. We quantized both weights and activations of the proposed network to 3 bits with high precision. In addition, we devised a dedicated hardware for the 3-bit quantized network. The feature is that forward propagation calculation is performed using only internal memory. As a result, data transfer with the outside of the chip, which requires a large amount of power, is eliminated. This circuit computes convolutions of 8 rows, 16 input channels, 16 output channels and 3 by 3 pixels in parallel. The convolution calculation performance at an operating frequency of 300 MHz is 11 TOPs/s. To our
knowledge, there is no report on the hardware implementation of a semantic segmentation network that has been quantized up to 3 bits.

The structure of this paper is as follows. Section 2 describes related researches. Section 3 describes the proposed 3-bit quantized network and describes the experimental results. Section 4 describes the hardware architecture based on this network. Section 5 describes the results of implementing the dedicated hardware on the FPGA. Section 6 concludes this work and talks future works.

2. Related works

Many works have already been done on low bit-width quantized CNNs, and the results were summarized in survey papers [1]. The inference accuracy of 8-bit quantization is almost the same as that of single-precision floating-point numbers, and it is known that the accuracy does not drop so much even if the bits are lower than 8-bit. Research on 1-bit quantization has been also in progress [2]. Development of dedicated hardware for the quantization network is in progress for the purpose of high power efficiency. Many of these studies target the category classification.

Although it is smaller in number than category classification, low-bit quantization of semantic segmentation networks has also been studied. In the literature [5], the network in which the convolution of the reconstruction part of FCN is replaced with residual blocks is quantized up to 2 bits with high accuracy. In the literature [11], authors developed a method called structure approximation, which transforms a floating-point network into a network aggregating a set of binary branches. The method could quantize a semantic segmentation network using dilated convolution with high accuracy.

There are some FPGAs for semantic segmentation. In the literature [6], authors implemented a semantic segmentation network on an FPGA using a technique called 8-bit two-hot quantization. In the paper [7], authors implemented an 8-bit quantization network with filter-wise pruning to reduce the computational complexity. Xilinx provides xdnn, an 8-bit inference CNN circuit [8]. It has also announced a 4-bit quantized circuit for inference, which is used to realize semantic segmentation [9].

3. 3-bit quantized CNN for semantic segmentation

3.1. Network structure

| No. | Name          | Input channels | Output channels |
|-----|---------------|----------------|-----------------|
| 1   | Enc_cbr0      | 3(12)          | 64              |
| 2   | Enc_down1     | 64             | 64              |
| 3   | Enc_cbr1      | 64             | 64              |
| 4   | Enc_cbr1_1    | 64             | 128             |
| 5   | Enc_down2     | 128            | 128             |
| 6   | Enc_cbr2      | 128            | 128             |
| 7   | Enc_cbr2_1    | 126            | 256             |
| 8   | Enc_down3     | 256            | 256             |
| 9   | Enc_cbr3      | 256            | 256             |
| 10  | Enc_cbr3_1    | 256            | 256             |
| 11  | Enc_down4     | 256            | 256             |
| 12  | Enc_cbr4      | 256            | 256             |
| 13  | Enc_cbr4_1    | 256            | 512             |
| 14  | Enc_down5     | 512            | 512             |
| 15  | Enc_cbr5      | 512            | 512             |
| 16  | Dec_up4       | 512            | 512             |
| 17  | Dec_cbr4      | 512            | 512             |
| 18  | Dec_cat4      | 512            | 512             |
| 19  | Dec_cbr4_1    | 768            | 256             |
| 20  | Dec_up3       | 256            | 256             |
| 21  | Dec_cbr3      | 256            | 256             |
| 22  | Dec_cat3      | 256            | 256             |
| 23  | Dec_cbr3_1    | 512            | 256             |
| 24  | Dec_up2       | 256            | 256             |
| 25  | Dec_cbr2      | 256            | 256             |
| 26  | Dec_cat2      | 256            | 256             |
| 27  | Dec_cbr2_1    | 384            | 128             |
| 28  | Dec_up1       | 128            | 128             |
| 29  | Dec_cbr1      | 128            | 128             |
| 30  | Dec_cat1      | 128            | 128             |
| 31  | Dec_cbr1_1    | 192            | 64              |
| 32  | Dec_up0       | 64             | 64              |
| 33  | Dec_conv      | 64             | 12              |

Table 1. Structure of CNN for semantic segmentation.
We adopted UNET as a neural network for semantic segmentation [10]. The UNET is an encoder-decoder architecture and features skip connections that connect encoders and decoders in the same layer. We modified the UNET for FPGA implementation. The network configuration is shown in Table 1. The layer name 'Enc' represents an encoder and 'Dec' represents a decoder. 'cbr' represents 3 by 3 pixels convolution, batch normalization, and ReLU processing in serial. 'down' represents max-pooling of 1/2 horizontal and vertical. 'up' represents the upsampling of the vertical and horizontal double by the bilinear interpolation. 'cat' represents a concatenation of the upsampled decoder output from the lower layer and the encoder output of the same layer. 'conv' represents a convolution of 3 by 3 pixels. The number added to the layer name indicates the level (resolution) of the layer after layer processing.

The computational complexity is decreased by reducing the number of convolutions of each layer from the original 3 times to 2 times. The memory usage is reduced by connecting the feature map to the decoder in the same layer after pooling instead of before pooling. To make the activation quantization easy, not only the encoder but also the decoder performs the batch normalization after the convolution. This network first performs a simple bilinear interpolation with twice the height and width, and then convolves 3 by 3 pixels to perform upsampling. Transposed convolution is common for upsampling. The transposed convolution requires mask processing because the position of the input pixels differ depending on the position of the target pixel. Furthermore, the accuracy of transposed convolution of kernel size 3 by 3 adopted in this study is not good. The combination of bilinear interpolation and convolution is simple, and it is unnecessary to change the convolution circuit.

3.2. Quantization method
The method of weight quantization will be described. In preliminary experiments, even if the upper and lower limits were set as the same, the semantic segmentation accuracy did not change. When the upper and lower limits of all convolution weights were set as the same and the network using floating-point numbers was learned, the standard deviations for all layers were about half of the upper and lower limits. So, we set the quantization range equal to the upper and lower limits. By making the quantization ranges of the weights of all layers the same, scaling between different layers becomes unnecessary. The bias of all convolutions is set to zero.

The activation quantization will be described. The output of the activation function, that is, the input of the next convolution is called activation. This network always performs batch normalization after convolution, and then activates with ReLU. The distribution of activations after batch normalization has a mean of 0 and a variance of 1. After that, since ReLU is performed, the activation value becomes 0 or more. Therefore, the quantization range of all activations are uniformly set to 0 or more and 2 or less.

The two quantization procedures are compared. Procedure 1 is the method of reference [5]. First, we learn a network where both weights and activations are floating-point numbers. Next, using this learning result (weights when the validation accuracy is the best) as initial values, 8-bit network is learned. After that, the low-bit quantization network is learned step by step while reducing both weights and the activations by 1 bit. The initial value of the weight is the learning result of the previous 1-bit more network. The procedure 2 sets the learning result of the floating-point number network as the initial value of the weight in the learning of all the quantization networks. In this procedure, the learning rate is set higher than in the procedure 1. In these two procedures, the method of [3] was adopted as the learning method of the quantization network. In this method, the quantized weight and activation are used in the forward propagation calculation, but the weight is stored as a floating-point number, and the gradient is calculated by the floating-point number in the back propagation calculation to update the weight.

3.3. Experiment
An experiment of network quantization was performed. First, the experimental method will be described. The semantic segmentation network based on UNET proposed in this section was used. The quantization range of weights for all convolutions was set to -0.0625 to 0.0625. The quantization range of all activations was set to 0 to 2. These ranges were determined by preliminary experiments.
The data set used was CamVid[12,13]. The Cambridge-driving Labeled Video Database (CamVid) is an on-vehicle image data set with a correct label for each pixel. The resolution was 480 by 352 pixels, 367 images were used for training, and 101 images were used for validation. The height was originally 360 pixels, but the upper and lower 4 pixels were deleted so that it would be repeatedly split by 2. There are 12 classes of labels, including others. Twelve kinds of classes are 'Sky', 'Building', 'Pole', 'Road', 'Pavement', 'Tree', 'SignSymbol', 'Fence', 'Car', 'Pedestrian', 'Bicyclist', 'Other'.

The batch size during learning was 8, the number of iterations was 15,000 for floating-point networks, and 10,000 for quantization networks. Adam was used for optimization. The initial learning rate was set to 0.001 for floating-point networks. In the case of the quantization procedure 1, the initial value of the learning rate was set to 0.0001 for the quantization network. In the case of procedure 2, it was set to 0.001. The learning rate was reduced to 1/10 for every 5,000 iterations. Cross entropy was used as the loss function. As augmentation, changes in hue, brightness, saturation, and left-right inversion were performed. We used PyTorch 0.4.1 as a machine learning framework [14]. We used pytorch-semsseg as the semantic segmentation network [15]. The CPU of the PC used was Intel Core i9-7900X 3.30GHz, memory 64GB, GPU was Nvidia GeForce 1080ti.

Mean IoU (Intersection over Union) was used as an evaluation index of accuracy. The mean IoU is expressed by the following formula.

\[
mean\ IoU = \frac{1}{N} \sum_{c=1}^{N} \frac{\text{area}_{c,\text{inf}} \cap \text{area}_{c,\text{gt}}}{\text{area}_{c,\text{inf}} \cup \text{area}_{c,\text{gt}}}
\]  

(1)

Here, \(N\) is the total number of classes, \(c\) is the class number, \(\text{area}_{c,\text{inf}}\) is the area of class \(c\) in the inferred label image, and \(\text{area}_{c,\text{gt}}\) is the area of class \(c\) in the ground truth label image. The denominator represents the union of \(\text{area}_{c,\text{inf}}\) and \(\text{area}_{c,\text{gt}}\), and the numerator represents the overlap (intersection) of the two areas.

![Image of test image, ground truth, inferred label images of fp32 and int3.](image1)

**Figure 1.** Test image, ground truth, inferred label images of fp32 and int3.

![Graph of quantization bits and mean IoU.](image2)

**Figure 2.** Quantization bits and mean IoU.
Figure 1 is a CamVid test image, a ground truth label image, a floating-point label image (fp32), and a 3-bit quantization label image (int3). It can be subjectively seen that the precision of fp32 and int3 does not change. Figure 2 represents the relationship between the number of quantization bits and Mean IoU. In both quantization procedures 1 and 2, Mean IoU dropped sharply at int2. The mean IoU of int3 in procedure 2 was 0.678, and the deterioration was slight as compared with 0.680 of the fp32. Comparing the two procedures, procedure 1 had a lower mean IoU. It was considered that in procedure 1, the learning rate for the quantized network was too low. However, if the value was increased, learning became unstable and the loss was increased.

4. Hardware architecture

Figure 3 shows the overall block diagram. FM is a Feature Memory and stores images or feature maps. WM is Weight Memory and stores the weight of convolution. PM is Parameter Memory and stores the parameters of batch normalization. Recent FPGAs have large-capacity SRAM inside, and we take this advantage. First, all weights and parameters are read from the external memory GM2 into the internal memories WM and PM. Then, all the input images are read from the external memory GM0 (DRAM) to the internal memory FM, the forward propagation calculation loop is repeatedly executed, and the final convolution calculation result is written to the external memory GM1. The fact that there is no data transfer with the external memory during the forward propagation is a characteristic of this architecture.

Figure 4. Detailed overall block diagram.
Figure 4 shows a detailed block diagram. IP performs bi-linear interpolation of twice the height and width. CV performs a convolution calculation below.

\[
u_{ijm} = \sum_{k=0}^{K-1} \sum_{p=0}^{H-1} \sum_{q=0}^{H-1} z_{i+p,j+q,k} h_{pqkm} + b_{ijm}
\]  

Here, \( u \) is output, \( z \) is input, \( h \) is filter coefficient, \( b \) is bias, \( i \) and \( j \) are coordinates in feature maps, \( p \) and \( q \) are coordinates in kernels, \( m \) is output channel number, \( k \) is input channel number, \( K \) is the number of input channels, \( H \) is kernel size. In this study, the kernel size is always three and the bias is always zero. BR performs batch normalization and ReLU. MP does max pooling. Each of these is divided into eight, and eight consecutive rows of the image or feature map are processed in parallel. The FM is composed of 10 banks, and 10 consecutive rows adding two rows above and below to the eight rows are read simultaneously. A corresponding write circuit WF and read circuit RF are connected to each FM. Since each CV exchanges data with all FMs, RB and WB on the way collect and deliver the data.

Each FM is a 48-bit word, and stores 16 consecutive 3-bit activations in one word. The same column of 10 consecutive rows is read from FM every cycle. RB reassembles this into eight data of three consecutive rows, and sends them to IP0 to IP7. Each IP combines three consecutive columns to generate 3 by 3 16-channel activations. 16 copies of this are sent to the corresponding CV. The CV is composed of 16 element circuits CVE, and processes 16 output channels in parallel. After that, the 16 output channels are processed by BR and MP and written to FM as one word.

Figure 5 shows a block diagram of the CVE. K00 to K15 perform 3 by 3 kernel calculations for 16 input channels. The result of the kernel calculation is given to the adder tree. The accumulator accumulates the product-sum operation results of all input channels for one output channel. To support unsigned 8-bit data in the input image, divide it into four 2-bits and append 0 to the beginning of each 2-bit to make four 3-bits. Then shifters must be inserted to the adder tree. Recent FPGA LUTs have 6 inputs. Since both activation and weight are 3-bit in this network, one digit of multiplication output can be realized by one LUT. In other words, multiplication of 3-bits can be realized with only 6 LUTs. In
addition, since the 6-input LUT can realize the 3-input addition for each digit by one LUT, the adder tree in the kernel can be efficiently implemented.

**Figure 6.** Timing diagram of semantic segmentation processor.

Figure 6 shows the timing diagram for the entire circuit. This is the case when the width of the input feature map is \(W\), the height is \(H\), and the channel is 64 for both input and output. This circuit is a pixel-by-pixel pipeline without stalls. The number of output channels that can be calculated simultaneously is 16, and all output channels are calculated every 16 output channels in sequence. The number of input channels that can be calculated simultaneously is 16, and all input channels are calculated every 16 input channels in sequence. After convolving all input channels for each output channel, then batch normalization is performed. When max pooling is not performed, 16 output channels are written as one word to FM. When performing max pooling, the maximum value is obtained by comparing with the result of the adjacent row every 2 columns (2 pixels). In this figure, the results of MP0 and MP1 are compared, and the maximum value is output from MP0. This circuit does not require a delay buffer of one row for max pooling.

**Figure 7.** Area reuse of feature memory.
This circuit calculates convolution of 3 by 3 pixels in 8 rows, 16 input channels and 16 output channels in parallel. When the operating frequency is $F = 300$ MHz, the number of operations (OPs) per second in the convolution calculation is $N = 11.059$ (TOPs/s). Here, one product-sum operation is divided into product and sum, and is counted as two operations. Batch normalization, ReLU, and max pooling operations are not included here.

Figure 7 shows the data allocation to FM. The numbers on the left side correspond to the layer numbers in Table 1. No. 0 is the initial state, and the input image IM before the start of processing is saved. No. 2 is the state after the processing of Enc_down1 is completed, and the output ED1 is saved. Since UNET has a skip connection between the encoder and the decoder, it is necessary to save the output of each encoder layer. No. 15 is the state after the encoding is completed, and ED1–ED5 are the outputs of each layer. Since the size of the feature maps EC1–EC5 that do not need to be saved is equal to or smaller than IM, the area allocated to IM is repeatedly used. Decoding processing is performed after No. 17. In decoding, the processing result of the lower layer is upsampled and convolved. The feature maps DC3–DC1 after upsampling reuse the ED4–ED2 and DC4–DC2 that have become unnecessary after the completion of the lower layer processing. After convolving the concatenations between ED4–ED1 and DC4–DC1, DC41–DC11 reuse the IM region. By reusing the area in this way, the FM size can be reduced.

5. FPGA implementation
We implemented the proposed UNET-based semantic segmentation processor on Alveo U200 from Xilinx. The Alveo U200 is an accelerator card involving an FPGA for image processing and AI inference, and it is used by connecting to the PCIe bus of a PC. The development environment was Vitis 2020.1 from Xilinx. The circuit scale of the proposed semantic division circuit was LUT 453,826/1,025,284, REG 685,525/2,135,017, BRAM 1,206/1,822, URAM 180/960, DSP 882/6,833, and it was estimated to operate at 300Hz.

Next, we compared with Vitis AI. Vitis AI consists of an FPGA-based DNN circuit developed by Xilinx, and a development environment such as quantization and compiler tools [16]. The neural network developed with machine learning frameworks such as Caffe and Tensorflow is quantized by 8 bits, compiled, and executed on the DNN circuit. When the original UNET was installed on the Alveo U200 using Vitis AI, the throughput for an image with a width of 256 pixels and a height of 256 pixels was 32 fps (frame per second). Equal to this study, if the UNET network configuration is made lighter and the resolution is the same as CamVid (480 × 352 pixels), it is estimated to be 30 fps. On the other hand, the throughput of the proposed circuit is 123 fps for CamVid. However, Vitis AI can also run a semantic segmentation network that requires less computation than UNET. On the other hand, looking at the results of implementing our circuit on U200, there is room in the circuit resources and there is room to increase the operating frequency, so there is a possibility that throughput performance of ours can be further increased several times.

6. Conclusion and future work
We improved the semantic segmentation CNN based on UNET for FPGA implementation, and quantized weights and activations up to 3 bits. The 3-bit quantized mean IoU, which represents the segmentation accuracy, was 0.678 for the CamVid data set, which was slightly less than the accuracy of floating-point numbers. Then, we devised a dedicated hardware architecture for the 3-bit quantized semantic segmentation CNN. The feature is that all calculations are performed using only the FPGA internal memory, which eliminates power consumption due to data transfer with external memory during the forward propagation calculation. This circuit is a pixel-by-pixel pipeline without stall, and performs 3 by 3 pixels convolution calculation of 8 rows, 16 input channels, and 16 output channels in parallel. The convolution calculation performance at an operating frequency of 300 MHz is 11 TOPs/s. It was implemented on Xilinx Alveo U200 and confirmed that it was possible to place and route. UNET can be used not only for semantic segmentation but also for motion estimation and disparity estimation.
Future work is to develop a CNN dedicated processor that performs semantic segmentation, motion estimation, and disparity estimation at the same time.

Acknowledgments
This work was supported by JSPS KAKENHI Grant Number JP 18K11350.

References
[1] ERWEI WANG, JAMES J. DAVIS, RUIZHE ZHAO, HO-CHEUNG NG, I, XINYU NIU, WAYNE LUK, PETER Y. K. CHEUNG, and GEORGE A. CONSTANTINIDES, “Deep Neural Network Approximation for Custom Hardware: Where We’ve Been, Where We’re Going”, ACM Computing Surveys, Vol. 52, No. 2, Article 40. Publication date: May 2019.
[2] Taylor Simons and Dah-Jye Lee, “A Review of Binarized Neural Networks”, MDPI Electronics 2019, 8, 661; doi:10.3390/electronics8060661.
[3] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, Yoshua Bengio, “Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 or -1”, arXiv:1602.02830 [cs.LG].
[4] Rastegari M., Ordonez V., Redmon J., Farhadi A., “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks”, In: Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, vol 9908. Springer, Cham. https://doi.org/10.1007/978-3-319-46493-0_32.
[5] He Wen and Shuchang Zhou and Zhe Liang and Yuxiang Zhang and Dieqiao Feng and Xinyu Zhou and Cong Yao, “Training Bit Fully Convolutional Network for Fast Semantic Segmentation”, arXiv:1612.00212 [cs.CV].
[6] Vogel, Sebastian and Springer, Jannik and Guntoro, Andre and Ascheid, Gerd, “Efficient Acceleration of CNNs for Semantic Segmentation on FPGAs”, Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, https://doi.org/10.1145/3289602.3294006.
[7] Shimoda M., Sada Y., Nakahara H., “Filter-Wise 7 Approach to FPGA Implementation of Fully Convolutional Network for Semantic Segmentation”, In: Hochberger C., Nelson B., Koch A., Woods R., Diniz P. (eds) Applied Reconfigurable Computing. ARC 2019. Lecture Notes in Computer Science, vol 11444. Springer, Cham. https://doi.org/10.1007/978-3-030-17227-5_26.
[8] Rahul Nimaiyar, Ashish Sirasao, “Xilinx ML suite”, IEEE Silicon Valley Machine Learning Compiler workshop- Fall 2018.
[9] “Convolutional Neural Network with INT4 Optimization on Xilinx Devices”, WP521 (v1.0.1) June 24, 2020.
[10] Ronneberger O., Fischer P., Brox T., “U-Net: Convolutional Networks for Biomedical Image Segmentation”, In: Navab N., Hornegger J., Wells W., Frangi A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI 2015. Lecture Notes in Computer Science, vol 9351. Springer, Cham. https://doi.org/10.1007/978-3-319-24574-4_28.
[11] B. Zhuang, C. Shen, M. Tan, L. Liu and I. Reid, "Structured Binary Neural Networks for Accurate Image Classification and Semantic Segmentation," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 2019. pp. 413-422, doi: 10.1109/CVPR.2019.00050.
[12] Brostow, Shotton, Fauqueur, Cipolla, “Segmentation and Recognition Using Structure from Motion Point Clouds”, ECCV 2008.
[13] Brostow, Fauqueur, Cipolla, “Semantic Object Classes in Video: A High-Definition Ground Truth Database”, Pattern Recognition Letters.
[14] https://pytorch.org/
[15] https://github.com/meetshah1995/pytorch-semseg
[16] https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html