Binarized Depthwise Separable Neural Network for Object Tracking in FPGA

Li Yang, Zhezhi He, Deliang Fan
Department of ECE, University of Central Florida, Orlando, FL
{liyang,elliot.he}@knights.ucf,dfan@ucf.edu

ABSTRACT

Object tracking has achieved great advances in the past few years and has been widely applied in vision-based application. Nowadays, deep convolutional neural network has taken an important role in object tracking tasks. However, its enormous model size and massive computation cost have became the main obstacle for deployment of such powerful algorithm in low power and resource limited embedded system, such as FPGA. Due to the popularization of the power-sensitive mobile platform, low power real-time tracking solution is strongly required. In order to address these challenges, we propose a low power and energy-efficient object tracking FPGA implementation based on a newly proposed binarized depthwise separable deep convolutional neural network. It can significantly reduce the model size and computation complexity simultaneously utilizing binarized (i.e., +1 and -1) depthwise separable convolution kernel and our proposed trainable threshold group binarization activation function. It can completely converts the dot product and accumulation based convolution operations into bit-wise XNOR and bit-count operations, while achieving state-of-the-art accuracy. Our proposed binarized depthwise separable model achieves ~57% Intersection over Union (IOU) on DJI object tracking dataset with only ~143.9Kb model parameter size. We then deploy our proposed model into the Xilinx PYNQ Z1 board with only 4.9Mb on-chip RAM. The experiment results show that our FPGA implementation achieves 11.1 frames per second for object tracking with only 2.61W.

ACM Reference Format:
Li Yang, Zhezhi He, Deliang Fan. 2019. Binarized Depthwise Separable Neural Network for Object Tracking in FPGA. In *Great Lakes Symposium on VLSI 2019(GLSVLSI'19), June 30–July 5, 2019, Tysons Corner, VA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145//3299874.3318034

1 INTRODUCTION

In the last couple years, the climate of Artificial Intelligence, especially deep learning, has swept the various domains owing to its prominent performance over traditional methods [7]. As of our interest, object tracking is a intriguing topic which is the core function of many real-world applications. In virtue of deep learning technique, a trained deep neural network (DNN) can achieve up to 100 frames per second [5]. Comparing with traditional object tracking algorithm, deep learning has more powerful ability to extract feature from data. However, it relies on massive operations and model size, which makes the mobile device deployment encounter great challenges in terms of both computation and storage capability. Recently, in DAC Design Contest 2018 [3], many low power FPGA object tracking designs have been proposed to address such high computation cost and memory usage.

The contributions of this work mainly consist of two folds: 1) In the algorithm perspective, we propose a new method to optimize YOLO based object tracking deep neural network simultaneously using approximate weight binarization, trainable threshold group binarization activation function and depthwise separable convolution methods, to greatly reduce computation complexity and model size; 2) In the hardware part, we propose a new corresponding deep neural network accelerator architecture in FPGA to reduce latency, communication between off-chip and on-chip memory, and to maximize throughput according to hardware resources.

2 BINARIZED DEPTHWISE SEPARABLE DEEP NEURAL NETWORK FOR OBJECT TRACKING

2.1 Depthwise Separable Convolution

Recently, the depthwise separable convolution [10] has been proven it's efficiency in many state-of-the-art deep neural networks, such as MobileNet [6] and BD-NET [4], which factorize a conventional convolution layer into a depthwise convolution and a convolution with 1x1 convolution called pointwise convolution.

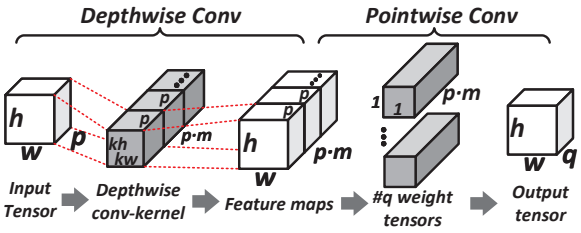

Figure 1: Data-flow for depthwise separable convolution layer. Normally, the channel expansion m is 1.[4]

The data flow of depthwise separable convolution is shown in fig. 1. In the depthwise convolution layer, each channel of input tensor performs convolution with $p \cdot m$ kernels in the size of $kh \times kw$ (i.e. kernel-height and width) correspondingly, which produces $p \cdot m$ feature maps. m is defined as *channel expansion*. Those generated feature maps are concatenated along its depth dimension as tensor in size of $h \times w \times (p \cdot m)$ which is taken as the input of pointwise convolution layer. The operation of such depthwise convolution

GLSVLSI '19, June 30–July 5, 2019, Tysons Corner, VA, USA

ACM ISBN 978-1-4503-6252-8/19/05…$15.00
https://doi.org/10.1145//3299874.3318034

can be mathematically described as:

$$F_{j\cdot \mathrm{p}+i} = X_i * W_{j\cdot \mathrm{p}+i},\ j \in [0, m-1] \tag{1}$$

where $i \in [1, \mathrm{p}]$ is the index of depth of input tensor $X \in \mathbb{R}^{h\times w\times p}$. On the contrary to the distinctive depthwise convolution, pointwise layer is just normal spatial-convolution layer with 1×1 convolution kernel size. The ratio of computational cost between depthwise separable convolution and conventional convolution can be calculated as:

$$\frac{h\cdot w\cdot kh\cdot kw\cdot p\cdot m + h\cdot w\cdot p\cdot m\cdot q}{h\cdot w\cdot kh\cdot kw\cdot p\cdot q} = \frac{m(kh\cdot kw+q)}{kh\cdot kw\cdot q} \tag{2}$$

such ratio is approximated to $m/(kh\cdot kw)$ when the number of output channels $q >> kh\cdot kw$. In our work, we set the channel expansion m to 1 and the kernel size $kh\times kw$ is 3×3. In this case, the computation cost is only 1/9 of the normal spatial-convolution.

Table 1: Architecture of Our Model

Layer	Filter Shape	Input Shape	Kbits
Conv1 dw	3x3x3	160x320	0.864
BinConv2	64x3x1x1	160x320	0.1875
MaxPool		80x160	
BinConv3 dw	3x3x64 dw	80x160	0.5625
BinConv4	128x64x1x1 dw	80x160	8
MaxPool		40x80	
BinConv5 dw	3x3x128 dw	40x80	1.125
BinConv6	128x128x1x1	40x80	16
MaxPool		20x40	
BinConv7 dw	3x3x128 dw	20x40	1.125
BinConv8	128x128x1x1	20x40	16
Conv9	25x128x1x1	20x40	100
Total			143.864
IOU (DJI)			57%

2.2 Our Model

2.2.1 Neural Network Structure. In this section, we first present our binarized depthwise separable convolution neural network for object tracking. Except that the first and last layers are real-value tensors, all the rest layers are binarized in both weight and activation. Conv9 shown in the table 1 is fully-connected layer for prediction. Inspired by YOLOv2[9], we use the anchor boxes method to predict bounding box location and confidence score to reflect how precise the bounding box is. In this work, we define confidence as IOU (Intersection over union) which is an evaluation metric that yielded by dividing the area of overlap by the area of union between the predicted bounding box and the ground-truth bounding box. We divide the input images into a $20x40$ grid, where each grid cell predicts 5 bounding boxes and computes confidence values for those boxes.

2.2.2 Weight Binarization. To binarize convolution kernel weights, we employ the popular approximation strategy for weight binarization first proposed in [8], which estimates the real-value weight kernel using a binary weight filter (i.e. -1 and +1) and a scaling factor E. The scaling factor should equal to the mean of absolute values of whole weight filters in current layer.

2.2.3 Trainable Threshold Group Binarization Activation Function and XNOR Convolution. In order to improve the accuracy while maintaining binary activation and XNOR based convolution operation (i.e. requiring both activation and weight in binary state), we propose a trainable threshold group binarization activation function, as shown in fig. 2, to replace original BinActive-XNORConv combination to a group of parallel BinActive-XNORConv modules, where each BinActive function has different trainable threshold v_k to generate binary x'_k from x.

$$x'_k = Sign(x - v_k) \tag{3}$$

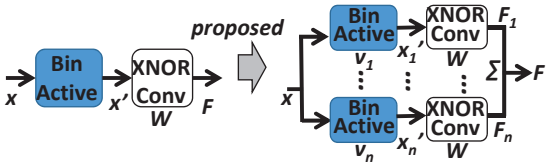

Figure 2: Proposed trainable threshold group binarization activation function

where trainable threshold v is indexed by k, and k is a integer ranging from 1 to n, where n=5 in this work. During the training of deep neural network, such threshold v_k is also considered as a trainable parameter similar kernel weight. However, during the training, the sign function in eq. (3) owns zero derivatives almost everywhere, which makes it impossible to calculate the gradient using chain rule in backward path (i.e. training phase). Thus, the Straight-Through Estimator (STE) [1, 2] is applied to calculate gradient in this work. In the backward path, the input gradient of binarization activation function clones the gradient at output, if it is in the range from -1 to +1. Otherwise, the gradient is cancelled to preserve training performance:

$$\text{Backward}: \frac{\partial g}{\partial x} = \begin{cases} \frac{\partial g}{\partial x'} & if\ |x| \le 1 \\ 0 & otherwise \end{cases} \tag{4}$$

During training, to keep the distribution of activation relatively stable, batch-normalization layer is inserted before binarization activation, which can normalize the feature maps to zero mean and unit variance. Moreover, to properly initialize v_k during training phase, we use a input sample to obtain the distribution of interlayer tensor then manually setup the initial values. It is worth to note that all the parallel XNORConv layer in each module share the same binary weights for XNOR based convolution computation. Therefore, our method doesn't increase the overall model size.

2.2.4 Binarized Depthwise Separable Convolution. In this subsection, we apply previously discussed approximation weight binarization and our proposed trainable threshold group binarization activation for both depthwise convolution and pointwise convolution. In this case, except the first and last layer, the weights of all layers are binarized with a scaling factor as well as activations.

The weight and activation binarization function can be replaced by *XNOR* operation.The computation of depthwise convolution can be reformatted as:

$$BinF_{j\cdot \mathrm{p}+i} = E_p * \sum_{k=1}^{n} 2*XNOR(X_i - v_{i,k}, W_{j\cdot \mathrm{p}+i}) - nN,\ j \in [0, m-1] \tag{5}$$

where N is the number of corresponding weight filters to create one output value which equals to $kh * kw$ in depthwise convolution.

2.2.5 Converting ReLU-BatchNorm-Binarization Activation-Weight Scaling Factor to Threshold Function. In this work, we propose to integrate weight scaling factor with ReLu-BatchNorm-Binarization activation, which could totally eliminate multiplication in our binarized depthwise separable convolution.

Batch Normalization layer can be considered an affine function $\boldsymbol{y} = k\boldsymbol{x} + b$. Therefore, the cascaded BatchNorm-Binarization Activation-Weight scaling factor can be described as:

$$\boldsymbol{y} = \begin{cases} \boldsymbol{E} * Sign(k\boldsymbol{x} + b) & if\ \boldsymbol{x} \geq 0 \\ b & otherwise \end{cases} \quad (6)$$

In the eq. (6), x means the output tensor $BinF$ from previous convolution layer according to eq. (5). E presents the weight scaling factor. In order to avoid multiplication and subtraction in eq. (5). We use x'' to represent n sum of the $XNOR(X, W)$, so the x can be described as:$\boldsymbol{x} = 2 * \boldsymbol{x''} - \boldsymbol{nN}$.

The eq. (6) can be realized through sign function ($Sign_\Delta()$) with adjusted threshold Δ, where $Sign_\Delta(x) = +1$ when $x \geq \Delta$ and $Sign_\Delta(x) = 0$ when $x < \Delta$. Therefore, eq. (6) can be rewritten as:

$$\boldsymbol{y} = \begin{cases} Sign_\Delta(x'') & if\ \boldsymbol{x''} \geq \boldsymbol{nN}/2 \\ b & otherwise \end{cases} \quad (7)$$

where $\Delta = -b/(Ek)$.

3 FPGA IMPLEMENTATION

3.1 Overall Architecture

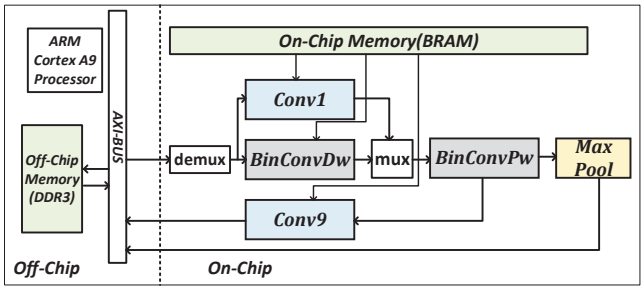

Figure 3: Overall Architecture

The overall architecture of our mapped model in FPGA is shown in fig. 3. There are totally four processing elements (PE) corresponding to four types of convolution layer respectively, namely Conv1, BinDw, BinPw and Conv9. Except first and last layer, all the rest binarized depthwise convolution layers reuse BinDw kernel which implement bitwise *XNOR* and *bitcount* operations. In order to further reduce communication between on-chip and off-chip, and maximize the usage of on-chip resources, BinDw and BinPw are allocated different hardware resources. In addition, it is worth to note that, benefiting from our deeply compressed model size (144Kbits), all of the network parameters can be stored in FPGA on-chip RAM. As widely known, communication between on-chip and off-chip memory is power hungry and time consuming. Our fully on-chip BNN could greatly reduce such communication and leaves more memory bandwidth for input images.

3.2 Hardware Optimization

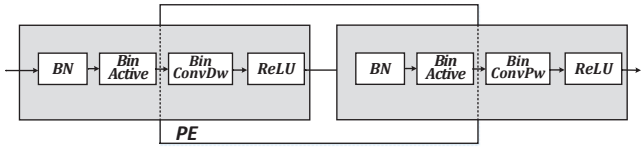

Figure 4: Hardware Block Architecture

3.2.1 Block Optimization. In order to more efficiently map model to FPGA, we further redefine the block architecture which makes it more hardware friendly without changing the whole pipeline architecture. fig. 4 shows the new processing element (PE) consisting of different computation components. Our optimized new PE consists of BinConv-ReLU and BatchNorm-BinActive. By redesigning PE in this way, it could be found that the outputs of each PE are in binary format since they are the direct outputs from binarized activation function (i.e. BinActive). Therefore, such optimization method guarantees inputs and outputs of each processing elements are both binary data, which greatly reduces the inter-layer communication data size. Moreover, such design is also necessary since we convert integrated ReLU-BarchNorm-BinActive into a more hardware friendly threshold function as described in eq. (7).

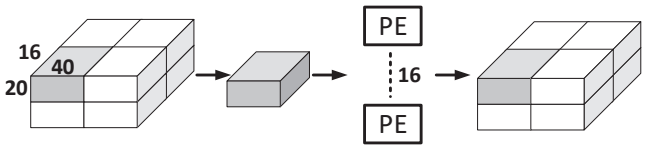

Figure 5: Feature Maps Group

3.2.2 Feature Map Group. Since the feature maps of different layers have different dimension, which means the computation pattern varies betwen layers. It is impossible for the whole feature maps to reuse *ConvDw* and *ConvPw* processing element. In order to solve this problem, we divide every feature map into the same size of groups as shown in fig. 5. Feature map groups are loaded into processing element for both depthwise and pointwise convolution, which will then create one corresponding output feature map group, one by one in pipeline.table 1 gives an example which using $20 \times 40 \times 16$ as the dimension of feature map groups.

3.2.3 Processing Element. The detailed processing element architecture is shown in fig. 6, which mainly consists of XNOR units, popcount units and following threshold units. As discussed in section 2.1, depthwise convolution and pointwise convolution have different computation cost, therefore the input buffer and XNOR unit contain different computation capacity. Processing elements are core computing unit to implement convolution operation. It can be seen that *XNOR-popcount-threshold* could compute in a parallel manner, which will load different weights(i.e. kernels) at the same clock cycle. Therefore, the number of parallel *XNOR* units, which is 16 in our work, corresponds to the output feature maps. As shown in fig. 7, depthwise and pointwise processing element can achieve 16×5 and $16 \times 16 \times 5$ computation in parallel respectively. Such operations could be easily implemented using LUTs in FPGA and are much more energy efficient compared to conventional dot-product, accumulation and Batch-Norm operations.

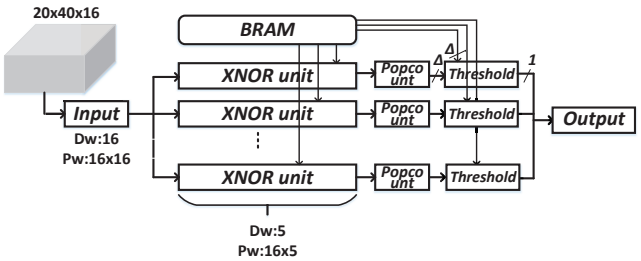

Figure 6: Processing Element Architecture

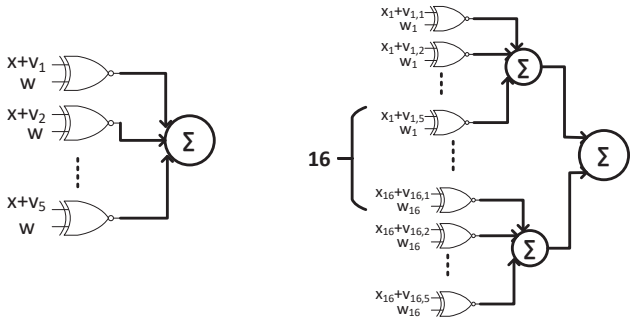

Figure 7: Multiple Shift XNOR for one input data in XNOR Unit, (a) Depthwise convolution, (b) Pointwise convolution

Table 2: COMPARISON WITH OTHER OBJECT TRACKING ON THE FPGA[3]

Name	IOU	Power	FPS	ES	TS
TGIIF	0.62	4.2	11.955	1.0318	1.2674
SystemsETHZ	0.49	2.45	25.968	1.3976	1.1794
iSmart2	0.57	2.59	7.349	1.0297	1.1636
traix	0.61	3.11	5.445	0.8869	1.1523
hwac-object-tracker	0.52	3.66	4.935	0.8155	0.932
Ours	0.57	2.61	11.1	1.1477	**1.224**

4 EXPERIMENTAL RESULTS

4.1 Experimental Results

To compare our work with other recent works. table 2 lists performance comparison with top 5 object tracking algorithms on DAC Design Contest 2018 [3] that all implement on the same PYNQ platform and test on the same DJI drone dataset. In order to evaluate the performance of different trackers, two evaluation metrics are proposed, *ES* and *TS*. *ES* means the energy consumption score which is mathematical described as: $1 + 0.2 \times \log_2(\frac{\bar{E}}{E})$, where $\bar{E}$ represents the average energy consumption of all trackers and E is the energy consumption for current work. *TS* is the final score which combines *IOU* and *ES* together, defining as: $IOU \times (1 + ES)$. It is obvious that our final TS score is ranked in the second place, which is only 0.04 smaller than TGIIF.

From pespective of model size and data type, TGIIF uses 8bit fixed point weight and activation. SystemsETHZ has full precision weights for the first and last layer, binary weights and 5 bit fixed point activation for the rest layers. 8bit fixed point weights and 16 bit fixed point activation are used in iSmart2. However, in our work, except the first and last layer use 32bit and 16bit fixed point weights and activations respectively, all the intermediate layers are binarized weights and activations (i.e. +1 and -1). Benefiting from depthwise separable convolution and trainable threshold group binarization activation, our model size is further compressed to 143.864$Kbits$ as shown in table 1. From computation complexity perspective, as discussed in section 3.2. We use five parallel bit-wise $XNOR$ and bit-count operations instead of multiplication and accumulation operations. In addition, taking advantage of depthwise separable convolution, the computation cost is only 1/9 of the normal spatial-convolution. On the performance side, The top 2 trackers have higher FPS (frame per second) than our work as shown in table 2. TGIIF divides whole workflow into multiple threads which can process multiple images in pipeline in the system level perspective. SystemsETHZ utilizes mini-batch images as the input of the hardware accelerator which also achieves multiple images processing in pipeline. In our future work, we will keep optimizing our speed in latency or FPS.

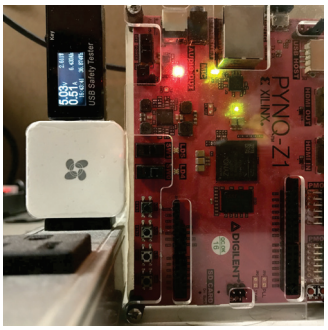

Figure 8: PYNQ Power Measurement

5 CONCLUSION

In this paper, we propose a new method to optimize YOLO based object tracking deep neural network simultaneously using approximate weight binarization, trainable threshold group binarization activation function and depthwise separable convolution methods, to greatly reduce computation complexity and model size. For FPGA hardware optimization, we propose a new corresponding deep neural network accelerator architecture in FPGA to reduce latency,communication between off-chip and on-chip memory, and to maximize throughput according to hardware resources. Our proposed model achieves~57% Intersection over Union (IOU) on DJI objecttracking dataset with only~143.9Kb model parameter size. We then deploy our model into the Xilinx PYNQ Z1 board with only 4.9Mb on-chip RAM. The experiment results show it achieves 11.1 frames per second for object tracking with only 2.61W.

ACKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation under Grant No. 1740126 and Semiconductor Research Corporation nCORE.

REFERENCES

[1] Yoshua Bengio et al. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).
[2] Matthieu Courbariaux et al. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).
[3] DAC. 2018. DAC Design Contest 2018. https://dac.com/content/2018-system-design-contest-1.
[4] Zhezhi He et al. 2018. BD-NET: A Multiplication-Less DNN with Binarized Depthwise Separable Convolution. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 130–135.
[5] David Held et al. 2016. Learning to track at 100 fps with deep regression networks. In *European Conference on Computer Vision*. Springer, 749–765.
[6] Andrew G Howard et al. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
[7] Yann LeCun et al. 2015. Deep learning. *nature* 521, 7553 (2015), 436.
[8] Mohammad Rastegari et al. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer, 525–542.
[9] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: better, faster, stronger. *arXiv preprint* (2017).
[10] Laurent Sifre et al. 2014. *Rigid-motion scattering for image classification*. Ph.D. Dissertation. Citeseer.