Low bit-based convolutional neural network for one-class object detection

Youngbin Kim,¹ Ouk Choi,² and Wonjun Hwang¹
¹Department of Artificial Intelligence, Ajou University, Suwon, Republic of Korea
²Department of Electronics Engineering, Incheon National University, Incheon, Republic of Korea

Low-performance systems such as mobile and embedded devices require an efficient deep neural network for object detection. In this letter, we propose a very efficient network made by both quantisation and model compression for detecting one class. First, our proposed network uses 1-bit weights to reduce the kernel parameter size and 8-bit activations to increase the speed. Second, we optimise the model size and computational power by compressing the maximum number of channels of the network. Therefore, compared to Darknet19, our proposed network infers 35 times faster on the CPU and saves over 7000 times memory. For fair evaluations, we built one-class object detection databases to detect subtitles of various videos and a specific class from the Pascal VOC database. We verify, compared to Darknet19 and Tiny you only look once (YOLO), that the proposed optimised network does not degrade in object detection accuracy with the efficient and applicable parameter sizes and computational complexity.

INTRODUCTION: Over the past few years, deep neural networks (DNNs) have achieved high performance in the field of computer vision. DNNs perform successfully in computer vision tasks, such as classification, segmentation, and detection, using convolutional neural networks (CNNs). With this success, many trials to apply DNN-based algorithms to consumer electronics (CE) devices are actively being conducted. However, CE devices have lower computational power and limited memory due to a price issue. Most DNN models are difficult to run on embedded systems and mobile applications. To solve this problem, many existing studies rely on low-bit quantisation [1–3], design compact networks [4, 5], and knowledge distillation [6, 7]. Among them, the low-bit quantisation and network architecture optimisation are good approaches for applying DNN models on the embedded systems, and there are studies to reduce the model size and computational complexity of DNNs. Using binary weights instead of full-precision weights directly reduces the memory size by about 32 times, and the multiplication operations in convolution are replaced with addition operations, increasing the speed by approximately twice. Even when only the weights are binarised, the surprising result is that the performance does not decrease for image classification and object detection. Moreover, designing a compact network is essential for more efficient inference for systems-on-chip (SoC) perspective.

For SoC-based applications, we develop an optimal deep learning model to detect one class by reducing the number of channels of the object detector as much as possible. Our main contributions are as follows: (1) We reduce the memory size by approximately 32 times using a 1-bit quantisation of the weight filter of the entire layer. In addition, we quantise full-precision activations to 8-bit activations. (2) We reduce the number of channels of the detector to 64 by applying the model compression method for optimal one-class object detection. (3) We verify the superiority of the proposed method using wild databases such as detecting subtitles at various videos.

Proposed method: A typical DNN architecture commonly consists of both 32-bit weights and 32-bit activation functions. It is difficult to build SoC targeting toward deep learning applications because of the large model size and high computational complexity. Our proposed network constructs 1-bit weights and 8-bit activations where we use the sign function to binarise the weight filter. The proposed quantisation network architecture is depicted in Figure 1. The i-th weight filter, \( W_i \), is represented to the binarisation, \( B_i \), by Equation (1):

\[
B_i = \text{sign}(W_i).
\]  (1)

![Fig 1](image)

During the backpropagation, the sign function is non-differentiable, and we use the straight-through estimator (STE) function [10] as shown in Equation (2). It preserves the gradient values and clips the gradient using a hard tanh function.

\[
\frac{\partial C}{\partial W} = \frac{\partial C}{\partial \text{STE}} \cdot W \quad \text{if } |W| \leq 1,
\]  (2)

where \( C \) is a cost function and the derivation \( \text{if } |W| \leq 1 \) is seen as propagating the gradient through hard tanh.

To make it easy to apply our proposed method to SoC, the activation function also performs 8-bit quantisation using TensorRT. It contains a real-valued scaling factor \( \alpha \) to approximate the FP32 value. The scaling factor \( \alpha \) is determined by minimising the Kullback–Leibler divergence between the FP32 and INT8 activations. We refer to this process as calibration within a quantisation workflow like in [8]. Note that our proposed network using 8-bit activations achieves a speed improvement of over 30%.

Algorithm 1 describes our proposed method for training a CNN using 1-bit weights and 8-bit activations. Each iteration of training a CNN involves two steps: Forward and backward propagations. In the forward propagation, we quantise only the activations and weights. However, to reduce the accuracy loss, the activations of the first and the last layers do not proceed to quantisation. In the case of weights, the memory size is minimised by quantising all the layers except the last layer. In the backward propagation, we simply train using STE.

![Algorithm 1](image)
posed method. We compared the results of Darknet19, Tiny YOLO, Tiny

tion and channel reduction method to Tiny YOLO to evaluate our pro-

Nvidia Titan Xp GPU and CPU, respectively. We applied the quantisa-

our codes based on Torchvision and PyTorch. We trained and tested on

imental environments are the same as YOLO9000 [9]. We implemented

fine-tuned it on our proposed model. Hyperparameters and other exper-

steps. We first trained the full-precision subtitle-pretrained model and

For all experiments, we trained the model in two

Experiment results: For all experiments, we trained the model in two

applications, but it is not easy to implement it directly for SoC. It is

largely because less working memory and algorithm complexity should

be preceded in order to implement the method on SoC. In this respect, we

use Tiny YOLO whose backbone architecture is a basic VGG network.

Compared to Darknet19, Tiny YOLO decreases the general performance

by 20%, but it is approximately four times faster. Furthermore, our

main target of this letter is the single class detection such as the subtitle

detection, and there is a chance to reduce the computational complexity

of our method for making an efficient detector. Specifically, as illustra-

ted in Figure 2(a), Tiny YOLO still has a maximum of 1024 channels

and consists of eight layers and has six max pooling, but our method

detects only one class to detect a single class on SoC, and it is possible

to reduce the total number of the used channels as much as possible.

As shown in Figure 2(b), our network architecture reduces the channel

size from the third convolutional layer to the last layer. For example,

the channel size of the last layer is reduced from 1024 to 64. Finally,

the memory size of the traditional Tiny YOLO was 43,896 kB, but we

can make our Tiny YOLO 64 with only 826 kB without much accuracy

degradation.

Dataset: Our subtitle dataset consists of images randomly collected

from Netflix or other video media for research purpose only. Aspect

ratios are different for all subtitle images. We make the aspect ratio

the same by padding zeroes at the bottom of all images. The subtitle dataset

consists of 9254 training images and 3100 test images with 480 × 288

pixels. We applied basic data augmentation for the subtitle dataset. For

example, we used random horizontal flipping, normalising, and random

cropping. Figure 3 shows example images of our subtitle dataset.

Table 1 presents the results of the subtitle dataset. To verify the ef-

fectiveness of the proposed network, the AP (average precision), model

size, and frame per second (FPS) were measured. Our model has an AP

reduction of approximately 2%, compared to the existing Darknet19 and

Tiny YOLO networks. However, our Tiny YOLO 64 model, which re-

duced the maximum number of channels of the network to 64, saved

approximately 30 times more memory compared to the Tiny YOLO net-

work. In this letter, we have further reduced the memory size by approx-

imately 30 times by quantifying the weight of the Tiny YOLO 64 model

by 1 bit. As a result of applying channel reduction and 1-bit weights,

our model size was 28 kB, making an efficient network for hardware

implementation.

From this result, we validate that our method is very efficient not

only in terms of memory size but also in inference FPS. Specifically, our

method achieved 94.04% and 96.23% with or without 8-bit activation,

only in terms of memory size but also in inference FPS. Specifically, our

method achieved 94.04% and 96.23% with or without 8-bit activation,

while the original Tiny YOLO showed 98.10%. From the viewpoint of

model size and FPS, the original Tiny YOLO has 43,896 kB and

while the original Tiny YOLO showed 98.10%. From the viewpoint of

model size and FPS, the original Tiny YOLO has 43,896 kB and

4.82 FPS, respectively, but our method has only 28 kB and its FPS is

4.977. Furthermore, our method has an AP reduction of approximately 2%,

compared to the existing Darknet19 and Tiny YOLO networks. However,

our Tiny YOLO 64 model, which reduced the maximum number of chan-

nels of the network to 64, saved approximately 30 times more memory

compared to the Tiny YOLO network. In this letter, we have further reduced

the memory size by approximately 30 times by quantifying the weight of

the Tiny YOLO 64 model by 1 bit. As a result of applying channel reduction

and 1-bit weights, our model size was 28 kB, making an efficient network

for hardware implementation.

YOLO 64, Tiny YOLO 64 with binary weights, and Tiny YOLO 64 with

binary weights and 8-bit activations. As presented in Tables 1 and 2, ap-

plying our proposed method gives a good result on the subtitle dataset

and PASCAL VOC bus dataset.

Table 1. Subtitle test detection results. We present average precision

(AP), model size, and frame per second (FPS). In all experiments,

input image size is 480 × 288. We measure the precision of subtitle

class when the intersection-over-union (IoU) of the bounding box is

0.5. We measure FPS on CPU

Table 2. PASCAL VOC 2012 bus class detection results. We present

AP (IoU = 0.5) results

![Image](image_url)
between Darknet19, Tiny YOLO 64 + 1-bit weights, and the proposed method. We verify that our method shows a speed of 35 times faster with a model size reduction of 70,000 times but with only approximately 7% accuracy reduction, compared with the original DarkNet19. This experiment demonstrated that our network is very efficient and suitable for detecting one class without significant accuracy degradation.

**Conclusion:** We study the use of an effective one-class detector. To design an efficient network, we constructed a model using 1-bit weights and 8-bit activations and compressed the model’s channels. According to our experimental results, our proposed network slightly decreases performance but significantly decreases memory size and significantly increases FPS.

**Acknowledgements:** This work was partially supported by MSIT support (IITP-2020-2018-0-01431) and by NRF-2020R1F1A1066049.

© 2021 The Authors. *Electronics Letters* published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

doi: 10.1049/ell2.12113

**References**

1 Wu, J., et al.: Quantized convolutional neural networks for mobile devices. In: Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA (2016)

2 Rastegari, M., et al.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: Proceedings of the European Conference on Computer Vision, Amsterdam, the Netherlands (2016)

3 Zhou, S., et al.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA (2016)

4 Zhang, X., et al.: Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA (2018)

5 Howard, A.G., et al.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861 (2017)

6 Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. In: Advances in Neural Information Processing Systems, Montreal, Canada (2014)

7 Zagoruyko, S., Komodakis, N.: Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In: International Conference on Learning Representations, Toulon, France (2017)

8 Nvidia: 8-bit inference with TensorRT. http://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inferencewith-tensorrt.pdf (2017)

9 Redmon, J., Farhadi, A.: Yolo9000: Better, faster, stronger. In: Computer Vision and Pattern Recognition, Honolulu, HI, USA (2017)

10 Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv:1308.3432 (2013)