IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 2 May 2022, accepted 29 June 2022, date of publication 4 July 2022, date of current version 11 July 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3188323

RESEARCH ARTICLE

Target Capacity Filter Pruning Method for Optimized Inference Time Based on YOLOv5 in Embedded Systems

JIHUN JEON[1], (Graduate Student Member, IEEE), JAEMYUNG KIM[1], (Graduate Student Member, IEEE), JIN-KU KANG[1], (Senior Member, IEEE), SUNGTAE MOON[2], AND YONGWOO KIM[3], (Member, IEEE)

[1]Department of Electrical and Computer Engineering, Inha University, Incheon 22212, South Korea
[2]Department of Computer Science and Engineering, Korea University of Technology and Education, Cheonan-si 31253, South Korea
[3]Department of System Semiconductor Engineering, Sangmyung University, Cheonan-si 31066, South Korea

Corresponding authors: Yongwoo Kim (yongwoo.kim@smu.ac.kr) and Sungtae Moon (stmoon@koreatech.ac.kr)

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2022R1G1A1007415). This paper was supported by the new professor research program of KOREATECH in 2022.

ABSTRACT Recently, convolutional neural networks (CNNs), which exhibit excellent performance in the field of computer vision, have been in the spotlight. However, as the networks become wider for higher accuracy, the number of parameters and the computational costs increase exponentially. Therefore, it is challenging to use deep learning networks in embedded environments with limited resources, computational performance, and power. Moreover, CNNs consume a great deal of time for inference. To solve this problem, we propose a practical method for filter pruning to provide an optimal network architecture for target capacity and inference acceleration. After revealing the correlation between the inference time and the FLOPs, we proposed a method to generate a network with the desired inference time. Various object detection datasets were used to evaluate the performance of the proposed filter pruning method. The inference time of the pruned network was measured and analyzed using the NVIDIA Jetson Xavier NX platform. As a result of pruning the number of parameters and FLOPs of the YOLOv5 network in the PASCAL VOC dataset by 30%, 40%, and 50%, the mAP decreased by 0.6%, 2.3%, and 2.9%, respectively, while the inference time was improved by 14.3%, 26.4%, and 34.5%, respectively.

INDEX TERMS CNN, filter pruning, FLOPs, inference time, YOLOv5.

I. INTRODUCTION

Recently, deep learning has shown excellent performance in various fields. Among the deep learning approaches, Convolution Neural Networks (CNNs) offers outstanding performance in the field of computer vision for such as image classification [1]–[3], object detection [4]–[10], and image segmentation [11]–[13]. This has been made possible by increasing the amount of data used and by improvement of hardware performance (by such as GPUs). However, as the networks become deeper and wider, the number of parameters and computational cost increase exponentially. Accordingly, networks consume more power for a large amount of computation and require long inference time and a large memory for a large number of parameters. In particular, this made CNNs difficult to use in embedded devices such as mobiles, autonomous vehicles, and drones, where resources are limited. To solve this problem, pruning is being actively researched [14]–[22] in a variety of studies on lightweight deep learning networks.

The associate editor coordinating the review of this manuscript and approving it for publication was Oguzhan Urhan.

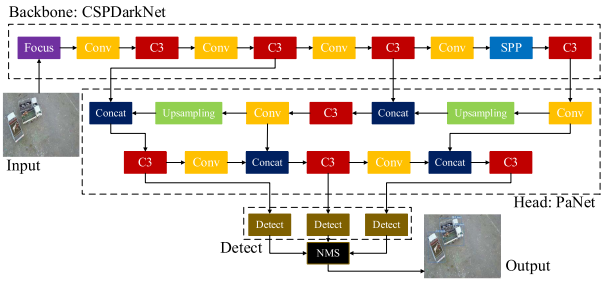


FIGURE 1. YOLOv5 architecture.

Pruning is a method that reduces the redundancy of a network according to a saliency score. This generates a network with fewer parameters requiring less computation than the baseline network. After training the network to figure out the importance of the parameters, unimportant parameters are removed according to their saliency scores by each algorithm. Moreover, pruning is divided into unstructured pruning [14] and structured pruning [15]–[22]. Unstructured pruning [14] judges the importance of each parameter individually and removes unnecessary parameters to obtain a sparse convolution structure. Structured pruning [15]–[22] removes unnecessary parameters by judging the importance of a bundle of several parameters. In this paper, by using structured pruning, YOLOv5 (an object detection network), is light-weighted to generate a network with a significantly reduced number of parameters and amount of computation. By measuring the lightweight network on the NVIDIA Jetson Xavier NX platform, it was confirmed that the inference time was improved. The contributions of this paper are as follows.

- Inspired by the holistic filter pruning (HFP) method proposed in [15], a network with a target number of parameters and FLOPs can be generated using the proposed target capacity filter pruning (TCFP). This was applied to the latest object detection network, YOLOv5, and the performance was evaluated on an edge device, NVIDIA Jetson Xavier NX.
- An optimal structure for network inference speed in the NVIDIA GPU was found experimentally and confirmed. Then, a practical network filter pruning framework was proposed and used to accelerate the network inference.
- The correlation between the amount of computation and the inference time was experimentally confirmed. Then, a method for generating a network with a target inference time using the new pruning method was proposed.

The structure of the remainder of this paper is as follows. In Section II, YOLOv5 [10] and related research on the pruning method are described. In Section III, we introduce target capacity filter pruning (TCFP), along with a practical network architecture optimization method for network acceleration on NVIDIA GPUs during pruning. In Section IV, quantitative experimental results of lightweight networks using TCFP are presented and analyzed. The correlation between the amount of computation and inference time, and a method for creating a network with target inference speed, is described. In Section V, the conclusions of this paper are discussed.

II. RELATED WORKS

This section describes the architecture of YOLOv5, the latest object detection network, and research related to pruning.

A. YOLOv5

Object detection refers to finding the locations and classes of specific objects in an input image. Object detection networks are divided into one-stage detectors and two-stage detectors. Moreover, the object detection process consists of region proposal and classification processes. Region proposal is the process of finding coordinates by specifying the locations of objects, and classification is the process of classifying the types of objects. In a one-stage detector, the region proposal stage and the classification stage are not divided but are performed at once, solving both stages simultaneously. Representative one-stage detectors include SSD [5] and YOLO [6]–[10]. A two-stage detector searches for objects by sequentially performing a region proposal and classification processes. First, the regions where the objects are likely to be found are determined through the region proposal process, and then the classification process is performed on the regions to actually detect the objects. The R-CNN [4] is a representative two-stage detector. The two-stage detector usually performs better than the one-stage detector. However, it has disadvantages: it is difficult to use for real-time object detection due to its slow inference speed, and its structure is complex and challenging to train.

In this paper, an accelerated object detection network is generated by applying TCFP to YOLOv5 [10], a state-of-the-art first-stage detector. YOLOv5 [10] consists of a backbone and a head module, as shown in Fig. 1. The backbone and head consist of the Conv module, Focus module, C3 module, and SPP module. In addition, there are several proposed versions of YOLOv5 [10] networks (small: s, medium: m, large: l, and xlarge: x), depending on the size of the network. In this paper, TCFP was applied to the Conv module, C3 module, and SPP module of a YOLOv5-large network. The Upsampling module and Concat module do not have parameters, so there was no need to apply the pruning method. The Focus and Detect modules have few parameters, and there is a risk that accuracy may decrease when a pruning method is applied, so the new pruning method was not applied.

B. PRUNING

The pruning process involves sparsity learning, pruning, and fine-tuning. The importance of artificial neural network connections is determined in the sparsity learning stage. In the pruning, which is based on the sparsity learned network, unimportant connections are determined according to their saliency scores, and the corresponding connections are removed. In the pruning process, the number of parameters and FLOPs of the network are reduced. The last process is the fine-tuning process to retrain the pruned network.

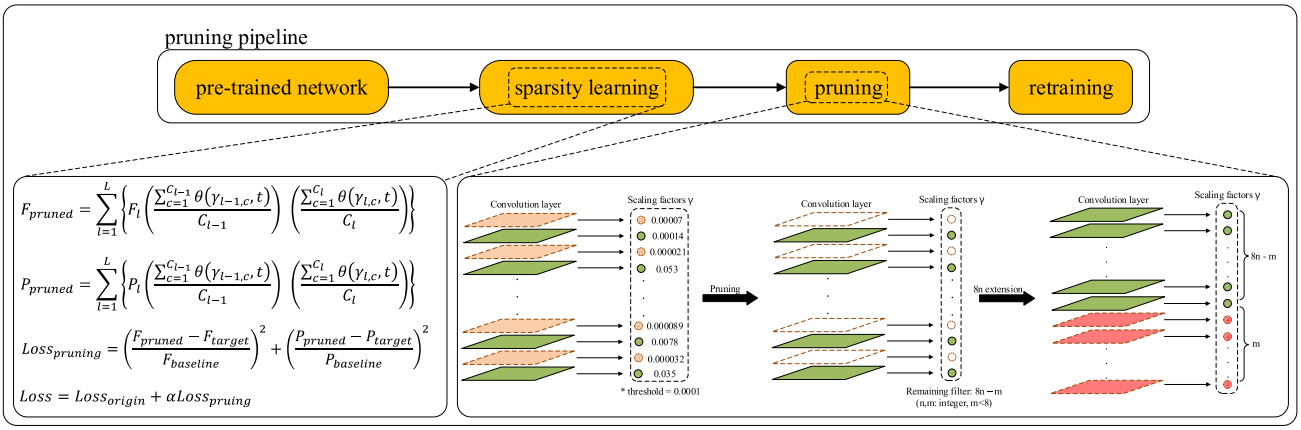


FIGURE 2. Target capacity filter pruning (TCFP) overview.

Pruning is classified as either unstructured/weight pruning [14] or structured pruning [15]–[22] according to the method of determining the importance of parameters and removing them. Unstructured pruning [14] removes parameters by determining the importance of each parameter according to the saliency score of each algorithm. Although it has the advantage of having a high compression ratio, many parameters are reduced in the fully connected layer. Therefore, it is inefficient in terms of the amount of computation because it does not effectively lighten the convolutional layer, where most of the network computations occur. Unstructured pruning also has the disadvantage of requiring specific library and hardware support for a sparse matrix.

In contrast, structured pruning [15]–[22] judges the importance of network connections according to the saliency score of each algorithm based on larger units such as channels. Structured pruning does not have a sparse matrix so that an existing library can be used, and memory usage can be reduced without requiring additional hardware. However, if a large number of parameters are pruned, the network performance may be degraded. Structured pruning methods [15]–[22] are divided into predefined pruning methods and automatic pruning methods according to the method of determining the pruning rate. The predefined methods [16]–[19] make the user define the pruning rate for each layer. However, the automatic pruning methods [15], [20]–[22] automatically pruned the entire network according to the algorithm used.

The pruning methods mentioned in the followings are the studies that used predefined pruning methods. Li *et al.* [16] used the L1-norm of the filter as a saliency score to judge the importance of the filter. Luo *et al.* [17] found and removed a filter with little effect by using the difference between the feature map before and after removing a specific channel of the filter. He *et al.* [18] used a soft pruning method to set a filter with a small L2-norm to zero every epoch and then retrained it. Lin *et al.* [19] found unimportant channels and their corresponding filters through their rank in the feature map and removed them. The predefined method has the disadvantage in that the user has to go through several trial-and-error processes to determine the pruning rate for each layer.

The automatic pruning methods can automatically create a pruned network architecture by an algorithm, so it requires less trial and error compared to the predefined pruning methods. Therefore, in this paper, we consider an automatic filter pruning method. Liu *et al.* [20] proposed network slimming, which removed the filter by judging the importance of a channel through the size of the scaling factor γ of the batch normalization layer. Li *et al.* [21] pointed out the problem of vanilla evaluation and used adaptive batch normalization to compare the performance of several networks pruned at a random rate for each layer and selected the one network structure with the highest performance. HFP [15] constructed a loss function using the scaling factor γ value of the batch normalization layer and used it to find the structure of the pruned network. Chen *et al.* [22] divide the parameter groups that affect the same output feature map into zero invariant groups (ZIG) sets, and then use the half-Space stochastic projected gradient (HSPG) optimizer to simultaneously perform sparsity learning and fine-tuning to remove unnecessary parameters. In network slimming [20], the number of remaining channels became zero after pruning, so a network could not be generated, and FLOPs could not be adjusted directly. Li *et al.* [21] may take a long time to find an optimal network due to a wide search space. Moreover, it may be challenging to generate a model with target parameters and target amount of computation. In HFP [15], an excessively pruned network could be generated beyond the target number of parameters and computational amount, after which it may take much time to find hyperparameters to generate a network that meets the target. Chen *et al.* [22] should use only the HSPG optimizer, and it is difficult to create a network with target parameters and FLOPs. In summary, other studies ([15], [20]–[22]) were a few or no mentions of improvement in inference speed due to the reduced amount of computation. Moreover, it is not possible or challenging to create a network with target

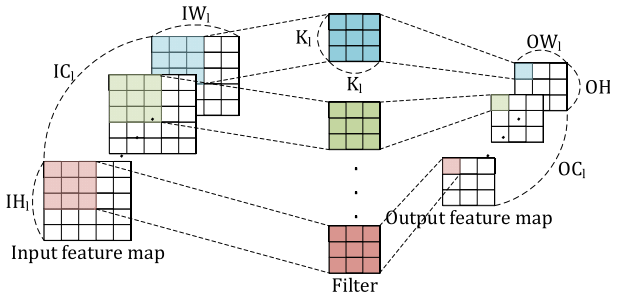


FIGURE 3. Operation on the convolution layer.

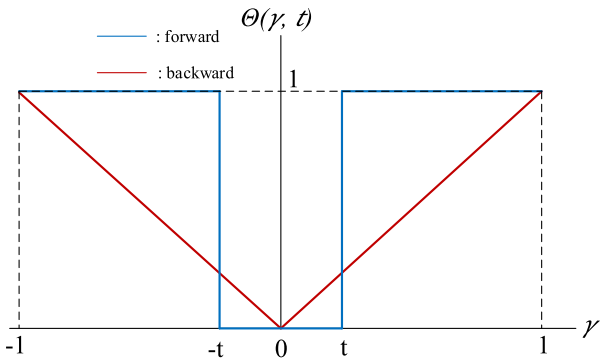


FIGURE 4. Indicator function.

parameters and FLOPs, and there is the limitation that most of them are applied and evaluated only for classification networks. However, in this paper, by applying the proposed pruning method to an object detection network, we not only create a network with target parameters and FLOPs but also improve the inference time in edge devices.

With the method reported in this paper, both parameters and FLOPs of a network can be pruned at the desired rate using TCFP, and we evaluated the performance of the proposed algorithm by comparing the network inference speed before and after pruning on the NVIDIA Jetson Xavier NX platform. In addition, the optimal structure for accelerating the inference speed in the NVIDIA GPU, and the correlation between inference speed and FLOPs, were confirmed experimentally. Therefore, using our proposed framework, a network with a specific amount of FLOPs can be generated, and a network with a target inference speed can easily be generated by predicting the inference speed.

III. PROPOSED METHODS

In this section, TCFP is explained in detail. The overall framework for our proposed method can be seen in Fig. 2. We perform sparsity learning based on a pre-trained network. In sparsity learning, the architecture of a lightweight network is determined that provides target numbers of parameters and computations. After that, pruning is performed using the size of the scaling factor γ of the batch-norm layer as a saliency score. During pruning, an 8n extension process is performed to generate a structure optimized for inference speed. Finally, based on the architecture generated by the pruning process, parameter values are initialized and retraining is performed.

A. AMOUNT OF COMPUTATION IN THE CONVOLUTION LAYER

The operation in the convolutional layer is performed as shown in Fig. 3. The terms IC_l, IH_l, and IW_l indicate the number of channels, height pixels, and width pixels of the input data of the l-th convolutional layer, respectively. K_l represents the number of pixels in the filter height and width of the l-th convolutional layer, and the number of channels in the filter is equal to the number of channels in the input. In addition, OC_l, OH_l, and OW_l indicate the number of channels, height pixels, and width pixels of the output data of the l-th convolutional layer, respectively. The number of height pixels and width pixels of the output data of the l-th convolutional layer follow (1) and (2). Here, S and P represent the stride and padding of the filter, respectively.

$$OW_l = \frac{IW_l + 2P_l - K_l}{S_l} + 1 \tag{1}$$

$$OH_l = \frac{IH_l + 2P_l - K_l}{S_l} + 1 \tag{2}$$

The amount of computation required to calculate one pixel of the output data is equal to the number of weights of one filter and can be expressed as (3). Moreover, to obtain all of the output data, (3) should be performed as many times as the number of pixels of the output data. Therefore, the amount of computation (number of calculations) in the l-th convolutional layer can be determined using (4).

$$F_{l,filter} = IC_l \times K_l \times K_l \tag{3}$$

$$F_l = F_{l,filter} \times OC_l \times OH_l \times OW_l \tag{4}$$

B. SPARSITY LEARNING

1) INDICATOR FUNCTION

In order to calculate the number of parameters and the amount of computation of the pruned network, an indicator function as in (5) was used. The filter to be pruned in the convolution layer is determined using the γ value (the scaling factor of the batch normalization layer) as the saliency score. In the batch normalization layer, the filters of the convolution layer that generate a channel affected by a value of γ smaller than the threshold t, can be pruned.

$$\theta(\gamma, t) = \begin{cases} 0, & if \ |\gamma| \le t \\ 1, & if \ |\gamma| > t \end{cases} \tag{5}$$

However, in the case of an indicator function such as (5), differentiation at the threshold value t is impossible, and because the differential value becomes zero at most γ, learning does not proceed appropriately during backpropagation. Therefore, a straight-through estimator (STE) [23] was used as the first derivative of the indicator function

TABLE 1. Comparison of YOLOv5 network performance according to conventional and proposed pruning methods for the PASCAL VOC dataset.

dataset	method	target pruning rate (Parameters/FLOPs)	mAP(0.5) (%)	mAP(0.5:0.95) (%)	parameters (remaining rate(%))	GFLOPs (remaining rate(%))	inference time (ms)
PASCAL VOC	Baseline	-	83.7	61.8	46,733,665	114.6	77.31
	Network Slimming[20]	30%/-	83.6	61.5	32,788,266 (70.2)	85.9 (75.0)	88.32
		40%/-	83.7	61.3	28,132,114 (60.2)	80 (69.8)	82.32
		50%/-	83.5	61.1	23,404,505 (20.1)	73.2 (63.9)	75.07
	HFP[15]	30%/30%	83.0	61.3	26,031,789 (55.7)	75.7 (66.1)	75.46
		40%/40%	80.1	57.1	17,858,088 (38.2)	46.5 (40.6)	64.53
		50%/50%	80.0	56.5	16,306,451 (34.9)	42.1 (36.7)	65.68
	Ours	30%/30%	83.1	61.1	32,924,033 (70.5)	81.9 (71.5)	66.24
		40%/40%	81.4	58.7	28,236,049 (60.4)	69.4 (60.6)	56.89
		50%/50%	80.8	57.9	23,722,113 (50.8)	61.7 (53.8)	50.66

during backpropagation to enable learning. The definition of STE is as follows:

$$\frac{\partial\theta(\gamma,t)}{\partial\gamma}=\begin{cases}-1, & if\ \gamma\le 0\\ 1, & if\ \gamma>0\end{cases} \tag{6}$$

The indication function in the forward and back-propagation process can be confirmed in Fig. 4.

2) LOSS FUNCTION FOR SPARSITY LEARNING

The use of CNNs in an embedded environment with limited resources is subject to several limitations. The limited memory makes it difficult to utilize an extensive network, and low computing power requires a long time for inference. Therefore, it is possible to generate a network with the number of target parameters and FLOPs using the proposed pruning method, which facilitates the use of CNNs in the embedded environment.

Equations (7) and (8) represent the amount of computation and the number of parameters of the network after pruning, as calculated using (5). The terms F_l, P_l, and C_l indicate the amount of computation, number of parameters, and number of channels of the l-th convolution layer, respectively. L indicates the number of layers in the network. The first parenthetical term on the right sides of (7) and, (8) indicates the effect of the filter pruned in the previous layer, and the second parenthetical term indicates the effect of the filter pruned in the current layer.

$$F_{pruned}=\sum_{l=1}^{L}\left\{F_l\left(\frac{\sum_{c=1}^{C_{l-1}}\theta\left(\gamma_{l-1,c},t\right)}{C_{l-1}}\right)\left(\frac{\sum_{c=1}^{C_l}\theta\left(\gamma_{l,c},t\right)}{C_l}\right)\right\} \tag{7}$$

$$P_{pruned}=\sum_{l=1}^{L}\left\{P_l\left(\frac{\sum_{c=1}^{C_{l-1}}\theta\left(\gamma_{l-1,c},t\right)}{C_{l-1}}\right)\left(\frac{\sum_{c=1}^{C_l}\theta\left(\gamma_{l,c},t\right)}{C_l}\right)\right\} \tag{8}$$

Equations (7) and (8) can be used to create a loss function that is the same as (9).

$$Loss_{pruning}=\left(\frac{F_{pruned}-F_{target}}{F_{baseline}}\right)^2+\left(\frac{P_{pruned}-P_{target}}{P_{baseline}}\right)^2 \tag{9}$$

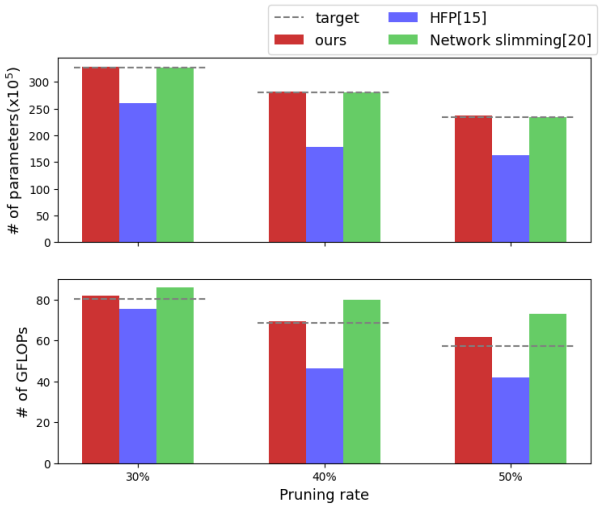
In (9), $F_{baseline}$ and $P_{baseline}$ represent the amount of computation and the number of parameters in the baseline network, and F_{target} and P_{target} represent the amount of computation and the number of parameters of the target pruned network, respectively. The terms F_{pruned} and P_{pruned} indicate the amount of computation and the number of network parameters remaining after removing unnecessary filters using the indicator function determined during sparsity learning. By combining the loss function $\mathrm{Loss}_{\mathrm{origin}}$ used for learning by the baseline network and the loss function $Loss_{pruning}$ used for pruning, as in (10), both can be solved simultaneously during training. Therefore, it is possible, after training is completed, to obtain an optimal network architecture with the target numbers of parameters and amount of computation.

$$Loss=Loss_{origin}+\alpha Loss_{pruing} \tag{10}$$

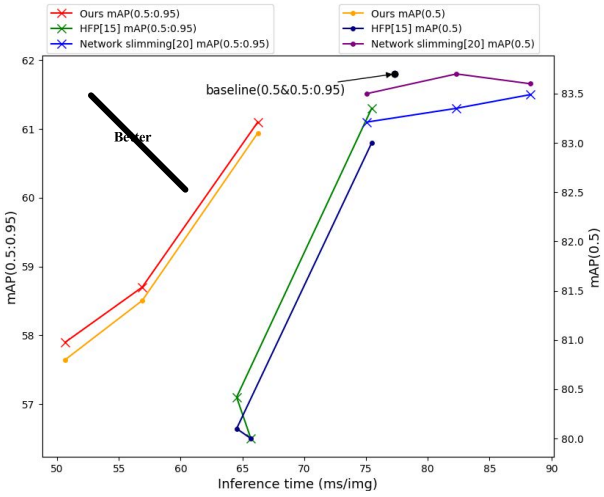
In (10), α is a scaling factor for balancing $Loss_{origin}$ and $Loss_{pruning}$. In this paper, half the size of $Loss_{origin}$ of the untrained network and the product of the maximum value of $Loss_{pruning}$ and α are set to be equal. For example, if the number of target parameters and the amount of computation of the network under sparsity learning are set to 50% of that of the existing network, the value of $Loss_{pruning}$ ranges from 0 to 0.5. Therefore, if the size of $Loss_{origin}$ of the untrained network is 1, the value of α becomes 1.

C. NETWORK ARCHITECTURE OPTIMIZATION PRUNING FOR INFERENCE SPEED

Most lightweight networks formed through pruning do not have a hardware-optimized architecture for CNN acceleration, such as a GPU. As a result, the inference speed of the pruned network may not be accelerated or may be relatively slow. We experimentally confirmed that the number of filters in the convolution layer should be a multiple of 8 to have an



(a) Number of parameters/FLOPs



(b) Pruned network performance

FIGURE 5. Experimental results from comparison of network performance (mAP), number of parameters, FLOPs, and inference time for different pruning methods using the PASCAL VOC dataset.

optimal architecture for inference speed on NVIDIA GPUs. Therefore, the number of filters was expanded to 8n based on the network architecture generated in the pruning step after sparsity learning. For example, if the number of filters in one convolution layer is 193, the number of filters is set to 200, and the 8n extension is made to be a multiple of 8. Through this practical method, network architecture optimization pruning for inference speed can easily be performed for NVIDIA GPUs.

D. CORRELATION BETWEEN INFERENCE TIME AND FLOPs

$$\frac{T^*}{T} \propto \sqrt{\frac{F^*}{F}} \tag{11}$$

Many users will expect network inference acceleration due to reduced amount of computation and reduced memory

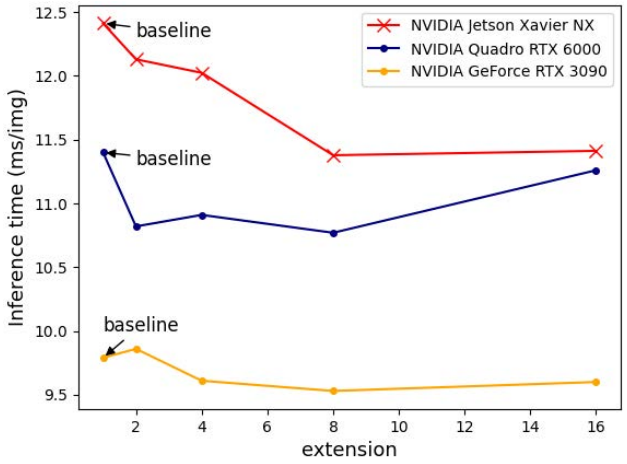


FIGURE 6. Comparison of YOLOv5 network inference times on different GPUs by extension size. The inference time on NVIDIA Jetson Xavier NX is plotted by dividing by 5.

usage through pruning. Therefore, if the correlation between the amount of computation and the inference speed can be identified, it is possible to determine in advance the FLOPs that the network must have to satisfy a specific inference time. Through TCFP, it is possible to generate a network with the target inference speed. Therefore, we found experimentally a correlation between inference time and the amount of computation. The relationship is not a linearly proportional relationship but has a non-linear correlation as in (11). F and F^* represent the FLOPs of the baseline network and the FLOPs of the pruned network, respectively. Moreover, T and T^* represent the inference time of the baseline network and the inference time of the pruned network, respectively. If the relationship between the amount of computation and the inference time as described above is used, it is possible to easily predict and generate a network with the target inference speed by using TCFP method.

IV. EXPERIMENTS

To evaluate our proposed TCFP method, three benchmark datasets for object detection were used: PASCAL VOC [24], VISDRONE [25], and COCO [26]. The parameter settings required for training and evaluation followed the default settings of YOLOv5 [10] except for batch size and image size. Inference time measurement was performed after fixing the batch size to 1. In this section, only the experimental results on the PASCAL VOC dataset are presented. The experimental results for the COCO and VISDRONE datasets are presented in the appendix.

A. COMPARISON RESULTS OF PERFORMANCE ACCORDING TO PRUNING METHODS

Table 1 shows the performance according to different pruning rates using three pruning methods for the PASCAL VOC dataset. The PASCAL VOC dataset includes VOC 2007 and VOC 2012. We conducted training using both versions of

TABLE 2. The ratios of inference time and FLOPs on 640 × 640 image in PASCAL VOC dataset.

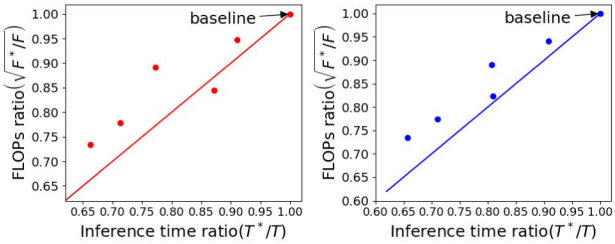
Pruning rate (Parameters/FLOPs)	GFLOPs	Network inference Time(ms)	Flops ratio	Inference time ratio
baseline	114.6	99.32	1	1
10%/10%	103.0	90.42	0.948	0.910
20%/20%	91.1	76.70	0.892	0.772
30%/30%	81.9	86.61	0.845	0.872
40%/40%	69.4	70.85	0.778	0.713
50%/50%	61.7	65.78	0.734	0.662

TABLE 3. The ratios of inference time and FLOPs on 1376 × 1376 image in VISDRONE dataset.

Pruning rate (Parameters/FLOPs)	GFLOPs	Network inference Time(ms)	Flops ratio	Inference time ratio
baseline	528.8	334.82	1	1
10%/10%	468.0	304.07	0.941	0.908
20%/20%	419.2	270.24	0.890	0.807
30%/30%	359.3	270.98	0.824	0.809
40%/40%	316.6	238.11	0.774	0.711
50%/50%	285.7	219.99	0.735	0.657

the dataset. It consists of about 16.5 k training images, 4.9 k validation images, and twenty types of objects. Each image has three channels and information about bounding boxes is provided in xml format. The batch size required for all network training, including sparsity learning, was fixed at 160. In addition, the image size parameter of YOLOv5 was set to 640 during training and testing. The experimental results are shown in Table 1. The first row of the table shows the performance of the unpruned baseline model. HFP [15] and our TCFP method was used to set the target pruning rate for parameters and FLOPs, but in the case of network slimming [20], this was not possible. Thus, pruning was performed by setting the target pruning rate based on the parameters.

Fig. 5(a) shows the number of parameters and FLOPs of the network generated according to each pruning method and rate. In the case of HFP [15], both parameters and FLOPs were excessively pruned compared to the target pruning rate. In the case of network slimming [20], parameters were appropriately pruned according to the target pruning rate, but the reduction in the amount of FLOPs was not large. In the case of our proposed pruning method, TCFP, both parameters and FLOPs were pruned to meet the target pruning rate, and a pruned network with the desired number of parameters and FLOPs was generated. Fig. 5 (b) shows the mAP performance versus the inference time of the pruned networks. For example, the leftmost point on the orange line represents the performance of a network with both parameters and FLOPs pruned by 50% using TCFP. The rightmost point represents the performance of a network pruned by 30%. The network pruned by HFP [15] has fewer FLOPs than the network pruned by the TCFP method, but requires a longer time for



(a) 640×640 PASCAL VOC dataset (b) 1376×1376 VisDrone dataset

FIGURE 7. Correlation between inference time and FLOPs. The Pearson correlation coefficient (PCC) of the (a) and (b) are 0.857 and 0.958, respectively.

inference and mAP also shows lower performance. In the case of the pruned network created using the network slimming method [20], the mAP performance is slightly higher, but this is a result of a large amount of difference in the FLOPs.

Moreover, when using network slimming [20], although FLOPs were reduced, the inference time was instead increased. Comparing the performance of the network in which parameters and FLOPs were pruned by 30% using our TCFP method, and the network in which only parameters were pruned by 30% using network slimming [20], the mAP(0.5) and mAP(0.5:0.95) are lower by 0.5% and 0.4%, but the inference time is 22.08 ms faster. Furthermore, in the case of using network slimming, the inference time deteriorates by 11.01 ms (14.2%), but improves by 11.07 ms (14.3%) with the TCFP method. By pruning the network using TCFP, the inference time could be improved by 14.3% to 34.5%.

B. NETWORK ARCHITECTURE OPTIMIZATION PRUNING ACCORDING TO EXTENSION AND GPU

The purpose of pruning is to generate a network with a small number of parameters and FLOPs to utilize a deep learning network in an environment with limited resources. Also, many users expect shorter inference times as FLOPs are reduced. However, there are cases in which the inference time does not decrease much or even increases more, such as with network slimming [20] and HFP [15] in Table 1. Therefore, we present a practical network architecture optimization pruning method for improving inference time. Most of the pruning process is performed after sparsity learning, as shown in Fig. 2. In the pruning process, the corresponding filters of the scaling factor having an absolute value smaller than the threshold are removed. After that, based on the architecture of the pruned network, it is possible to improve the inference speed of the pruned network effectively and in a practical way. Using the 8n extension improves the architecture so that the number of all filters is a multiple of 8, and the proposed 8n extension technique can easily be applied to other pruning algorithms.

To evaluate the performance of the proposed 8n extension, we measured and compared the inference time according to the extension size using an NVIDIA GeForce RTX 3090, Quadro RTX 6000, and Jetson Xavier NX. Fig. 6 showed the

TABLE 4. Comparison of YOLOv5 network performance according to conventional and proposed pruning methods for the VISDRONE dataset.

Dataset	Method	Target pruning rate (Parameters/FLOPs)	mAP(0.5) (%)	mAP(0.5:0.95) (%)	Parameters (remaining rate(%))	GFLOPs (remaining rate(%))	Inference Time (ms)
VISDRONE	Baseline	-	45.4	26.7	46,679,815	528.8	212.12
	Network Slimming[20]	30%/-	43.9	25.5	32,374,561(69.4)	351.0(66.4)	203.48
		40%/-	43.8	25.5	27,481,430(58.8)	323.7(61.2)	192.09
		50%/-	44.1	25.6	22,915,355(49.1)	299.2(56.6)	182.03
	HFP[15]	30%/30%	44.2	25.7	26,550,988(56.9)	322.1(60.9)	169.96
		40%/40%	44.2	25.7	20,934,154(44.8)	275.6(52.1)	162.60
		50%/50%	44.3	25.5	20,116,552(43.1)	266.8(50.5)	156.99
	Ours	30%/30%	44.7	26.1	31,486,695(67.5)	359.3(67.9)	179.80
		40%/40%	44.2	25.7	27,285,879(58.5)	316.6(59.9)	147.71
		50%/50%	44.3	25.8	22,787,767(48.8)	285.7(54.0)	137.01

change in inference time according to the extension size when inference involving the PASCAL VOC dataset was performed using the network in which the parameters and FLOPs were pruned by 40% using the TCFP. In all environments, when the number of filters was extended in a multiple of 8, the shortest inference time was required. In addition, the NVIDIA Jetson Xavier NX, GeForce RTX 3090, and Quadro RTX 6000 have Volta architecture, Ampere architecture, and Turing architecture, respectively. The versatility of the 8n extension was confirmed by showing that the inference time was improved with all three architectures.

C. NETWORK INFERENCE TIME VS PRUNING RATE

The inference time measurement reported in this subsection was performed for two cases with input image sizes of 640 × 640 and 1376 × 1376. In the case of a 640 × 640 input image, the network was trained and pruned using the PASCAL dataset, and in the case of an input image of 1376 × 1376 size, the network was trained and pruned using the VisDrone dataset. The performance of the VisDrone dataset is presented in the appendix. Tables 2 and 3 list the experimental results shown graphically in Fig. 7, where the correlation between inference time and FLOPs is also shown. It can be seen that the pattern of change of the inference time ratio and the FLOPs ratio according to the pruning rate is similar. The error between the inference time ratio and the FLOPs ratio is tiny, so it will be possible to predict effectively the inference time of the network before pruning. Moreover, when the correlation between the inference time and FLOPs data were quantitatively analyzed through Pearson correlation coefficient (PCC) [27], the PCC in the Pascal VOC dataset was 0.857 and the PCC in the VisDrone dataset was 0.958.

Both values are more significant than 0.85, indicating a strong correlation between FLOPs and inference time [28]. If the correlation between inference time and FLOPs is used, it will be possible to predict and generate a network simply with the target inference time.

V. CONCLUSION

In this paper, we proposed a target capacity filter pruning (TCFP) able to improve the inference speed of object detection networks. The proposed method makes it possible to create a network with an architecture optimized for the desired inference speed and with the targeted number of parameters and FLOPs. In addition, the correlation between inference time and FLOPs according to the pruning ratio was confirmed through various experiments. As a result, we confirmed the performance of the proposed filter pruning method using various object detection datasets and the YOLOv5 network.

APPENDIX

Tables 4 and 5 show the performance according to the various pruning ratios using three different pruning algorithms for the VisDrone and COCO datasets, respectively. The parameter settings required for training and evaluation followed the default settings of the YOLOv5 [10] network except for batch size and image size. The inference time measurement was carried out by fixing the batch size to 1.

A. VISDRONE DATASET

The VisDrone-2019 dataset consists of 6471 training images and 1610 test images, with 10 types of objects. Each image has 3 channels, and information about bounding boxes is provided in the form of text files. The batch size required for all training of the network, including sparsity learning, is fixed at 32. Also, the image size parameter of YOLOv5 [10] was set to 1376. Table 4 shows the experimental result. In the case of HFP [15], both parameters and FLOPs were over-pruned compared to the target pruning rate. As a result, the mAP showed lower performance than TCFP and required a longer inference time. In the case of network slimming [20], it was possible to create a network with the target number of parameters. Although it is not possible to set the desired ratio for pruning of FLOPs, as a result of pruning, the networks with an amount similar to the FLOPs targeted in this paper were generated. However, the mAP degradation is the most severe compared to other methods, and the improvement in inference time is insignificant compared to other methods. In the case of our TCFP method, both the number of parameters and FLOPs were pruned to match the target pruning rate. The inference speed and mAP performance were also

TABLE 5. Comparison of YOLOv5 network performance according to conventional and proposed pruning methods for the COCO dataset.

Dataset	Method	Target pruning rate (Parameters/FLOPs)	mAP(0.5) (%)	mAP(0.5:0.95) (%)	Parameters (remaining rate(%))	GFLOPs (remaining rate(%))	Inference Time (ms)
COCO	Baseline	-	65.6	46.1	47,056,765	115.6	77.89
	Network Slimming[20]	30%/-	64.9	45.4	32,950,356(70.0)	96.8(83.7)	88.51
		40%/-	64.6	45.2	28,209,018(59.9)	89.2(77.2)	82.98
		50%/-	63.8	44.3	23,552,898(50.1)	79.5(68.8)	75.26
	HFP[15]	30%/30%	64.8	45.3	25,430,652(54.0)	84.0(72.7)	84.72
		40%/40%	63.1	43.6	16,090,426(34.2)	67.6(58.5)	71.19
		50%/50%	63.5	43.4	12,361,007(26.3)	57.2(49.5)	67.83
	Ours	30%/30%	64.1	44.8	33,346,653(70.9)	82.2(71.1)	64.57
		40%/40%	63.6	44.2	28,729,341(61.1)	72.8(63.0)	63.62
		50%/50%	61.8	42.7	24,079,533(51.2)	61.9(53.5)	54.96

the highest. The network with both parameters and the FLOPs pruned by 50% using the TCFP method has fewer parameters and FLOPs than the network pruned by 30% using HFP [15]. In addition, mAP(0.5) and mAP(0.5:0.95) are 0.1% higher and the inference time is also 32.95 ms shorter. In addition, the network with both parameters and the FLOPs pruned by 50% using TCFP has fewer parameters and the FLOPs than the network pruned by 30% using network slimming [20]. However, mAP(0.5) and mAP(0.5:0.95) are higher by 0.4% and 0.3%, respectively. The inference time is also shorter by 66.47 ms.

B. COCO DATASET

The COCO dataset consists of approximately 118 k training images, 5 k validation images, and 80 types of objects. Each image has 3 channels, and information about the bounding boxes is provided in the form of a json file. The batch size required for all training of the network, including sparsity learning, was fixed at 128. Moreover, the image size parameter of YOLOv5 [10] was set to 640. Table 5 shows the experimental results. In the case of HFP [15], FLOPs are pruned close to the desired rate, but the number of parameters is excessively pruned compared to the target pruning rate. In the case of network slimming [20], the number of parameters can be pruned as much as desired, but the FLOPs are not significantly reduced compared to other methods. In the case of our proposed TCFP method, both the number of parameters and FLOPs are pruned close to the target pruning rate. In addition, when the network is pruned by other methods, the inference speed is rather slow or hardly improved, but in the case of TCFP, it can be confirmed that the inference speed is significantly improved. The inference speed of the network in which parameters and FLOPs are pruned by 30% through TCFP is 3.26 ms shorter than in the network in which parameters and FLOPs are pruned by 50% using HFP [15]. Moreover, mAP(0.5) and mAP(0.5:0.95) is 0.6% and 1.4% higher, respectively. When compared with the network in which parameters and FLOPs are pruned by 30% using network slimming [20], mAP(0.5) and mAP(0.5:0.95) are 0.7% and 0.5% lower, respectively, but the inference time is 23.96 ms shorter.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, San Diego, CA, USA, 2015, pp. 1–14.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.
[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. CVPR*, Jun. 2015, pp. 1–9.
[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.
[5] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 21–37.
[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 779–788.
[7] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 7263–7271.
[8] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.
[9] A. Bochkovskiy, C.-Y. Wang, and H.-Y. Mark Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*.
[10] G. Jocher, *YOLOv5*. Accessed: Jan. 5, 2021. [Online]. Available: https://github.com/ultralytics/yolov5
[11] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 3431–3440.
[12] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput.-Assist. Intervent.* Cham, Switzerland: Springer, 2015, pp. 234–241.
[13] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2881–2890.
[14] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1–9.
[15] L. Enderich, F. Timm, and W. Burgard, "Holistic filter pruning for efficient deep neural networks," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2021, pp. 2596–2605.
[16] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*.
[17] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, Oct. 2017, pp. 5058–5066.
[18] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," 2018, *arXiv:1808.06866*.
[19] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "HRank: Filter pruning using high-rank feature map," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2020, pp. 1529–1538.

[20] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, Oct. 2017, pp. 2736–2744.
[21] B. Li, B. Wu, J. Su, G. Wang, and L. Lin, "EagleEye: Fast sub-net evaluation for efficient neural network pruning," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2020, pp. 639–654.
[22] T. Chen, B. Ji, T. Ding, B. Fang, G. Wang, Z. Zhu, L. Liang, Y. Shi, S. Yi, and X. Tu, "Only train once: A one-shot neural network training and pruning framework," 2021, *arXiv:2107.07467*.
[23] G. Hinton, "Neural networks for machine learning," Coursera, Video Lectures, 2012.
[24] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes challenge: A retrospective," *Int. J. Comput. Vis.*, vol. 111, no. 1, pp. 98–136, Jan. 2015.
[25] P. Zhu, L. Wen, D. Du, X. Bian, H. Fan, Q. Hu, and H. Ling, "Detection and tracking meet drones challenge," 2020, *arXiv:2001.06303*.
[26] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.* Zurich, Switzerland: Springer, 2014, pp. 740–755.
[27] H. E. Soper, A. W. Young, B. M. Cave, A. Lee, and K. Pearson, "On the distribution of the correlation coefficient in small samples. Appendix II to the papers of 'student' and RA Fisher," *Biometrika*, vol. 11, no. 4, May 1917, pp. 328–413.
[28] P. Schober, C. Boer, and L. A. Schwarte, "Correlation coefficients: Appropriate use and interpretation," *Anesthesia Analgesia*, vol. 126, no. 5, pp. 1763–1768, May 2018.

JIHUN JEON (Graduate Student Member, IEEE) received the B.S. degree in electronics engineering from Inha University, where he is currently pursuing the M.S. degree in electrical and computer engineering. His research interests include deep learning, computer vision, and systems-on-chip design.

JAEMYUNG KIM (Graduate Student Member, IEEE) received the B.S. degree in electronics engineering from Inha University, where he is currently pursuing the Ph.D. degree in electrical and computer engineering. His research interests include systems-on-chip design, field-programmable gate array-based design, and convolutional neural networks optimization and acceleration.

JIN-KU KANG (Senior Member, IEEE) received the B.S. degree from Seoul National University, Seoul, South Korea, in 1983, the M.S. degree in electrical engineering from the New Jersey Institute of Technology, Newark, NJ, USA, in 1990, and the Ph.D. degree in electrical and computer engineering from North Carolina State University, Raleigh, NC, USA, in 1996. From 1983 to 1988, he was with Samsung Electronics Inc., South Korea, where he was involved in memory design. In 1988, he was with Texas Instruments, South Korea. From 1996 to 1997, he was with Intel Corporation, Portland, OR, USA, as a Senior Design Engineer, where he was involved in high-speed I/O and timing circuits for processors. Since 1997, he has been with Inha University, Incheon, South Korea, where he is currently a Professor and leads the System IC Design Laboratory, Department of Electronics Engineering. His research interest includes high-speed/low-power mixed-mode circuit design for high-speed serial interfaces.

SUNGTAE MOON received the B.S. degree from Chonnam National University, Gwangju, South Korea, in 2005, the M.S. degree from the Gwangju Institute of Science and Technology (GIST), Gwangju, in 2007, and the Ph.D. degree in aeronautical engineering from KAIST, in 2021. From 2007 to 2010, he was with the Agency for Defense Development (ADD), where he developed a mission computer for aircraft. From 2011 to 2012, he was with the National Security Research Institute (NSRI) and worked in the area of security in embedded systems. From 2012 to 2022, he was a Senior Researcher with the Artificial Intelligence Research Division of the Korea Aerospace Research Institute (KARI). Since 2022, he has been with the Korea University of Technology and Education (KOREATECH), Cheonan-si, South Korea, where he is currently an Assistant Professor with the School of Computer Science and Engineering. His current research interests include drone swarm flight systems, navigation algorithms, and object detection based on deep learning.

YONGWOO KIM (Member, IEEE) received the B.S. and M.S. degrees from Inha University, Incheon, South Korea, in 2007 and 2009, respectively, and the Ph.D. degree in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2019. From 2009 to 2017, he was a Senior Engineer with Silicon Works Company Ltd., Daejeon. From 2019 to 2020, he was a Senior Researcher with the Artificial Intelligence Research Division, Korea Aerospace Research Institute, Daejeon. Since 2020, he has been with Sangmyung University, Cheonan-si, South Korea, where he is currently an Assistant Professor with the Department of System Semiconductor Engineering. His current research interests include image/video processing algorithms, super-resolution, and deep learning hardware architecture for vision processing.