IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 May 2023, accepted 24 May 2023, date of publication 29 May 2023, date of current version 5 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3280552

RESEARCH ARTICLE

CP-CNN: Computational Parallelization of CNN-Based Object Detectors in Heterogeneous Embedded Systems for Autonomous Driving

DAYOUNG CHUN[1], (Graduate Student Member, IEEE), JIWOONG CHOI[2], HYUK-JAE LEE[1], (Member, IEEE), AND HYUN KIM[3], (Senior Member, IEEE)

[1]Inter-University Semiconductor Research Center, Department of Electrical and Computer Engineering, Seoul National University, Seoul 08826, South Korea
[2]NVIDIA, Santa Clara, CA 95051, USA
[3]Department of Electrical and Information Engineering, Research Center for Electrical and Information Technology, Seoul National University of Science and Technology, Seoul 01811, South Korea

Corresponding author: Hyun Kim (hyunkim@seoultech.ac.kr)

This work was supported by the Research Program funded by the Seoul National University of Science and Technology (SeoulTech).

ABSTRACT The success of research using convolutional neural network (CNN)-based camera sensor processing for autonomous driving has accelerated the development of autonomous driving vehicles. Since autonomous driving algorithms require high-performance computing for fast and accurate perception, a heterogeneous embedded platform consisting of a graphics processing unit (GPU) and a power-efficient dedicated deep learning accelerator (DLA) has been developed to efficiently implement deep learning algorithms in limited hardware environments. However, because the hardware utilization of these platforms remains low, performance differences such as processing speed and power efficiency between the heterogeneous platform and an embedded platform with only GPUs remain insignificant. To address this problem, this paper proposes an optimization technique that fully utilizes the available hardware resources in heterogeneous embedded platforms using parallel processing on DLA and GPU. Our proposed power-efficient network inference method improves processing speed without losing accuracy based on analyzing the problems encountered when dividing the networks between DLA and GPU for parallel processing. Moreover, the high compatibility of the proposed method is demonstrated by applying the proposed method to various CNN-based object detectors. The experimental results show that the proposed method increases the processing speed by 77.8%, 75.6%, and 55.2% and improves the power efficiency by 84%, 75.9%, and 62.3% on YOLOv3, SSD, and YOLOv5 networks, respectively, without an accuracy penalty.

INDEX TERMS Autonomous vehicle, convolutional neural network, embedded platform, low-power design, parallel processing, real-time system.

I. INTRODUCTION

Deep learning algorithms have delivered outstanding performance in various fields, and the use of these algorithms in autonomous vehicles has been actively studied. Unlike in other fields, autonomous driving applications need to consider three main features: high accuracy, real-time operation, and power efficiency [1]. Failure to guarantee these three features reduces the efficiency of autonomous driving and can result in dangerous accidents. The general structure of the autonomous driving algorithm processing is to perform the perception process of each sensor input (e.g., camera, LiDAR, and RADAR) independently and then integrate each output for planning and control processing [2], [3]. It is noteworthy

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

that the camera-based object detection has higher latency than the other sensors (*i.e.*, LiDAR and RADAR); thus, fast processing of camera input is essential to synchronously integrate the detection results of the camera sensor with those of other sensors [2], [4]. Furthermore, low-power systems are essential for autonomous driving platforms because autonomous vehicles generally operate on batteries with a limited energy budget. Accordingly, perception, planning, and control algorithms for autonomous driving are typically performed on embedded systems [5].

In recent years, to take advantage of both graphics processing unit (GPU)-based and field programmable gate array (FPGA)-based platforms, the embedded GPU board, which is widely used as an autonomous driving platform, is composed of a heterogeneous system that contains a deep learning accelerator (DLA) as well as a central processing unit (CPU) and GPU [6], [7]. DLA has been proposed for deep learning inference in embedded systems because it is more power-efficient than GPU and provides a solution for operating deep learning applications on mobile devices, such as low-power edge-computing [8], [9], [10], [11]. Consequently, heterogeneous embedded systems equipped with DLA are expected to be used continuously in various deep learning applications. However, because DLA has limited computational performance compared with GPU, it is difficult to use DLA alone for running autonomous driving algorithms that utilize a complex network structure with high computational cost and memory access to achieve high accuracy [12], [13], [14], [15]. Therefore, finding the optimal operating method for a heterogeneous embedded system considering the three features required for autonomous driving applications (*i.e.*, high accuracy, real-time operation, and power efficiency) is challenging but highly important.

Most studies aiming to accelerate deep learning inference on existing embedded systems [16], [17], [18], [19], [20], [21], [22], [23] have used the CPU and GPU environment as the main platform without including DLA. For example, [23] aims to achieve system speed improvement through CPU/GPU pipelining in multiple embedded devices. In particular, they focus on reducing the latency of each layer and consequently, some studies propose lightweight techniques such as quantization and pruning, which can reduce the network size but result in a loss of accuracy [24], [25], [26], [27]. Other studies [28], [29], [30], [31], [32], [33] have attempted to optimize convolutional neural networks (CNNs) in terms of their processing speed and power efficiency within limited resource environments, but they also adversely affect accuracy. These studies, accompanied by a reduction in accuracy, are unsuitable for autonomous driving applications, where high accuracy is essential. Some deep learning solutions that have recently been proposed for autonomous driving [1], [34], [35] are relatively accurate with an acceptable processing speed. However, because they are verified in GPU-based server environments, they are not guaranteed to achieve the same processing speed in real autonomous vehicles based on an embedded environment. Therefore, it is necessary to optimize autonomous driving applications by increasing processing speed and power efficiency without loss of accuracy in an embedded system.

To achieve this goal, this paper proposes a parallelization-based optimization method that fully utilizes hardware resources to perform CNNs in heterogeneous embedded systems. GPU offers high computing speed but low power efficiency, whereas DLA achieves a higher power efficiency but is computationally slower than GPU. Considering these characteristics, we accelerate the processing speed of existing deep learning algorithms and reduce power consumption without loss of accuracy by enabling DLA and GPU operations to be processed in parallel. It is noteworthy that a study using both DLA and GPU together in heterogeneous systems has not yet been reported. The division of the network between DLA and GPU must consider network structure, layer dependencies, device utilization time, and data-transfer time. Disregarding layer dependencies results in faulty inference. Likewise, failure to consider the device utilization and data transfer time would not allow the proposed method to be considered as a network inference operation optimized for embedded systems. Based on these constraints, the proposed method finds the optimal network partitioning point in the heterogeneous system to fully utilize DLA and GPU. Consequently, it achieves the most efficient parallel processing by attaining the best trade-off between processing speed and power efficiency. The experimental results show that the proposed method increases the processing speed by 77.8% and power efficiency by 84% on YOLOv3 [36] without loss of accuracy. Moreover, the proposed technique is highly compatible and can be applied to various CNN-based object detectors such as YOLOv3, SSD [37], and YOLOv5 [38].

II. BACKGROUND

A. OPTIMIZATION OF DEEP LEARNING INFERENCE ON EMBEDDED PLATFORMS

In general, the inference phase in deep learning proceeds as follows [17]: (1) pre-processing, (2) network operation, and (3) post-processing. Recently, deep neural networks (DNNs) have become very complex; the network operation has thus become overwhelmingly large, and consequently, the processing is mainly performed on GPU. The computational complexity of a network operation depends on the network architecture and input size. In particular, for applications requiring high accuracy, such as autonomous driving, the inference time increases because a deep and complex network structure is used [1], [39], [40]. Typically, pre-processing and post-processing times are shorter than the time required for the network operation [5] and can be hidden by parallelization on the CPU. Therefore, the network operation time on GPUs determines the processing speed of deep learning algorithms. Hence, methods to reduce GPU operation time are actively studied to accelerate deep learning algorithms on embedded systems.

ENet [28] proposes an architecture that quickly processes the semantic segmentation algorithm for autonomous driving in embedded platforms. ENet attempts to solve the low processing speed of the existing semantic segmentation algorithm with a large kernel size using small kernels (*e.g.*, 3×3 and 1×1). In addition, the feature map resolution is scaled to reduce the model size and the number of operations significantly. Consequently, ENet improves the processing speed of embedded systems. FRDet [41], an embedded platform-based object detection algorithm targeting autonomous driving, proposes a model lightweight method using a fire-residual module for Gausisan YOLOv3 [1]. By combining the residual skip connection [42] and fire module [43], FRDet effectively reduces the model size and processing speed while minimizing the accuracy loss. Joint optimization [29] aimed at optimizing the object detection algorithm Tiny-YOLO [44] in an embedded system focuses on processing speed and power efficiency. In this study, tucker decomposition [45] and 16-bit quantization are used to reduce the network operation, thereby reducing the processing time. In addition, CPU and GPU frequencies are experimentally selected to optimize the energy consumption per processing speed. This method reduces the computation required for the decomposition process and enables fast processing. MobileNet [30] is an architecture designed for the fast processing of edge devices by reducing the number of parameters using channel reduction, depth-wise separable convolution, and inverted residuals. Tiny-SSD [46] proposes network architecture for an embedded environment by combining the feature extraction structure of SSD [37] with the fire module proposed in SqueezeNet [43]. DF-SSD [32] proposes a DepthFire module that efficiently increases processing speed by integrating depth-wise convolution structure [30] and fire module [43]. These schemes reduce the network size, and thus, the processing speed increases while achieving high accuracy compared to Tiny-YOLOv3. However, all the methods mentioned above are accompanied by a loss of accuracy. For example, the ImageNet accuracy of MobileNetv1, MobileNetv2, and SqueezeNet is 70.6%, 72.0% and 60.4%, respectively, which is more than 4% of accuracy drop compared to the accuracy of ResNet50 (=75.9%) [30], [42], [43], [47]. As such, most approaches that directly reduce the amount of network computation based on changes in the network architecture lead to loss of accuracy; hence, they are not suitable for autonomous driving applications where high accuracy must be guaranteed.

In addition to algorithm-level implementations, various studies have also been conducted to optimize pre-trained model inference at the level of the hardware system to prevent such accuracy degradation. For example, nvDLA [6] is an accelerator proposed by NVIDIA as a solution for deep learning inference. To improve the power efficiency and processing speed, nvDLA [6] uses a special-purpose hardware engine optimized for deep learning inference and analyzes the memory access patterns that would need to be parallelized during deep learning inference. Nevertheless, nvDLA [6] delivers 4.1 TFLOPS in Float16 (FP16) operation, which provides insufficient computing power compared to GPU. In addition, layer functions that can be operated in DLA do not yet operate optimally. In other words, because the layer functions supported by DLA are limited to general layer functions such as convolution layers, it is not easy to fully process various networks in DLA. TensorRT (TRT) [48] is an optimization toolkit for deep learning models that includes an optimizer and a runtime engine. The model graph is simplified by optimizing DNN operations via quantization, bit-precision, and layer/tensor fusion in a pre-trained model. The processing speed is also improved by creating a runtime environment suitable for the platform and hardware architecture.

B. OPTIMIZATION OF DEEP LEARNING INFERENCE ON HETEROGENEOUS SYSTEMS

Efficient processing of network inference using heterogeneous processors has been studied [16], [17], [18], [21], [22], [49]. MSCDNN [16] performs runtime scheduling to increase throughput using a multi-phase processing method that uses a different input for each device in a heterogeneous system (data parallelism). However, this method is not suitable for autonomous driving applications, which require processing real-time streaming images, where the network needs to use the first-in-first-out approach for processing. DeepX [17] splits the layers into the CPU and GPU using a deep architecture decomposition process at the framework stage to perform low-latency executions of deep networks (layer parallelism). μLayer [18] proposes a channel-wise distribution between GPU and CPU by dividing the layers at a ratio that minimizes the latency (model parallelism). Yolobile [21] proposes to allocate a branch with low latency to the CPU for parallel processing with GPU in the branch structure. However, most of these methods focus on reducing the layer-level latency and are proposed along with other latency reduction methods (*i.e.*, pruning [17], [27] and quantization [14], [18], [50]), resulting in a loss of accuracy. In addition, the scope of these methods for efficiently dividing tasks and utilizing devices in a heterogeneous system has been limited to CPU and GPU devices.

Other studies on heterogeneous parallel processing are not limited to CPUs and GPUs [51], [52]. Rodriguez-Borbon et al. [51] use a field-programmable gate array (FPGA) along with CPUs and GPUs to partition and pipeline applications, resulting in an increased processing speed and energy efficiency compared to GPU homogeneous system. Yang et al. [52] propose a heterogeneous parallel processor containing many processing elements to increase processing speed by parallelizing multiple tasks. However, because these studies only target simple computational tasks, they are not suitable for application to DNN operations with a large amount of computation and data transfer. In conclusion, these previously proposed solutions cannot optimally utilize state-of-the-art (SOTA) heterogeneous systems containing DLAs for DNN processing.

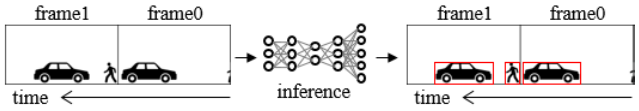

FIGURE 1. Streaming input processing in autonomous driving applications.

III. PROPOSED METHOD

This section proposes the method to parallelize the CNN on a DLA and GPU in a heterogeneous embedded system. The objective of this method is to improve processing speed and power efficiency without loss of accuracy while maintaining the order in which the streaming input arrives. As shown in Fig. 1, input images are generated in chronological order in autonomous driving applications. If the image order is not maintained, the detected position of the object will not match the real-time driving situation. Therefore, it is more important to process real-time image input from the camera sequentially than to process multiple images simultaneously. Based on an analysis of problems that occur during network partitioning, an optimized parallelization method for embedded systems is proposed considering the trade-off between processing speed and power efficiency.

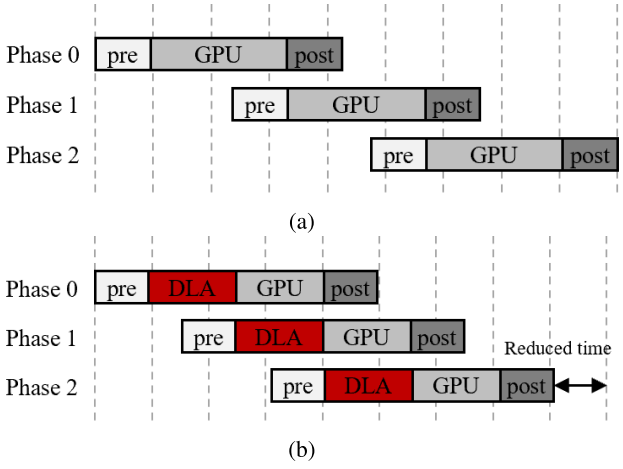

FIGURE 2. An example of a time diagram of inference phase in convolutional neural networks to show that processing times can be reduced by using the GPU and DLA together rather than using only the GPU alone: (a) GPU system, (b) DLA and GPU system.

A. PARALLELIZATION

In the case of network inference on existing GPU-only platforms, even when the CPU pre-processing and post-processing steps are hidden, network inference of the next phase is possible only after finishing the GPU operation of the previous phase, as shown in Fig. 2(a). Hiding CPU processing does not significantly improve the speed of network inference because the time required for GPU operation is much longer than pre-/post-processing on the CPU [5]. Parallel processing within GPU cannot reduce GPU operation time because the CNN operation is usually a feed-forward operation that computes the layers sequentially, which means that dependencies exist between the layers. In other words, if the layers with dependencies are processed in parallel, the feed-forward operation will be disrupted, and the intended results will not be obtained. Therefore, even in a heterogeneous system equipped with DLA and GPU, it is impossible to process the layers in one phase in parallel. However, parallel processing between different phases is possible if the dependency of the multi-phase layer operations is observed. Based on this observation, concurrent multiple-phase processing is possible by allowing parallel DLA and GPU processing to maximize hardware utilization in a heterogeneous system that includes both DLA and GPU. As shown in Fig. 2(b), when DLA and GPU process in parallel, the operation is hidden by the overlapping time between these devices, which reduces the total inference time and increases the throughput of the system. Since this method fully utilizes hardware resources without changing the network structure, there is no loss of accuracy. Accordingly, it is noteworthy that the proposed method is compatible with several baseline networks as well as several existing lightweight techniques that change the network structure or operation methods, including approximation studies with insignificant loss [53], [54], [55], without any dependency. In addition, the proposed method is advantageous in terms of power efficiency and processing speed because operations that were entirely performed on GPU (with its poor power efficiency) are divided and computed on both DLA (which is more efficient with respect to power consumption) and GPU.

B. OPTIMIZATION OF PARTITIONING POINTS

To process one network in parallel on DLA and GPU, selecting a network partitioning point (PP) is necessary. As shown in Fig. 3(a), it is important to select a network PP that fully utilizes both devices by considering the difference in computational speed between DLA and GPU. As shown in Fig. 3(b), when the operation time of DLA is shorter than that of GPU, DLA remains idle during GPU operation. In contrast, as shown in Fig. 3(c), when the operation time of DLA is longer than that of GPU, GPU experiences idle time during DLA operation. This device's idle time, caused by not considering the utilization time of each device, leads to the system performance degradation. Therefore, when setting the network PP, device dependency must be considered to ensure that both devices are fully utilized without idle time.

After selecting the appropriate network PP, the data transfer time between the two devices must also be considered. The data transfer time depends on the layer dependency according to the PP and the number of partitions. The layer dependency depends on the network structure of each model; this is described in Section III-C, along with the process of applying the proposed technique. Regarding the number of partitions, an additional device-to-device memory copy (memcpy) occurs between the two devices,

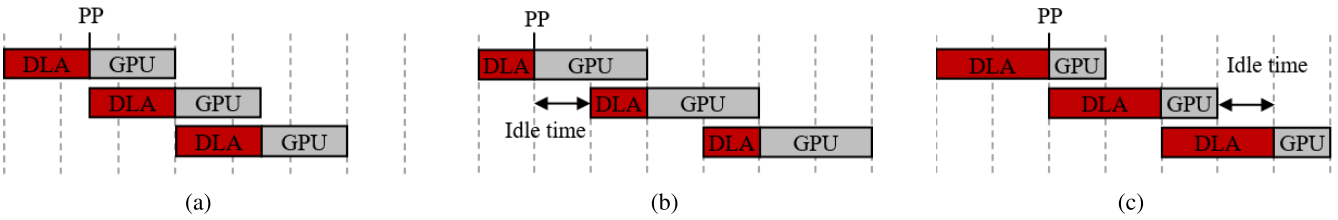

FIGURE 3. An example of time diagrams of device utilization according to the layer division point for device execution time: (a) DLA = GPU, (b) DLA < GPU, (c) DLA > GPU.

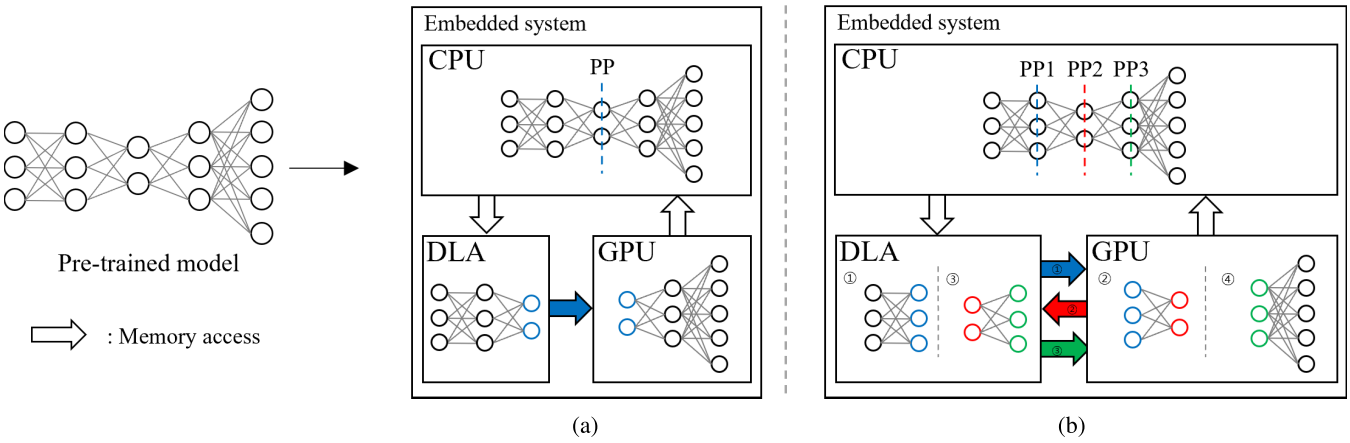

FIGURE 4. Embedded system process according to the network partitioning: (a) 2-partition structure, (b) 4-partition structure.

DLA and GPU, for each increased partition point, as the number of partitions increases. This is because in the case of the conventional inference using only GPU, only memcpy of CPU to GPU (MemCpyHtoD) for the input image and GPU to CPU (MemCpyDtoH) for the output results, but the proposed method using additional DLA essentially includes memcpy for inter layer output sharing between DLA and GPU (MemCpyDtoD). For example, on a platform with one DLA and one GPU, when the network is divided into two partitions, one device switch occurs for inference, as shown in Fig. 4(a) (*i.e.*, blue arrow). However, as shown in Fig. 4(b), when the network is divided into four partitions and the inference is processed in parallel by GPU and DLA, three times as many memory accesses are required to transmit the layer output between two devices. A simple comparison of the 2-partition structure and 4-partition structure without considering the data transfer time may indicate that the overall processing speed is higher in the 4-partition structure because more detailed and practical parallel processing is possible by dividing the network into smaller layers. However, since the overhead of the data transfer time also affects the inference time, these constraints should be comprehensively considered. Additionally, if the network is divided into smaller layers and processed in parallel, each device must simultaneously process multiple phases. In general, GPU in an embedded platform is limited in terms of its ability to handle multiple kernel operations simultaneously [56], [57]; thus, considering the processability depending on hardware resources is also important. Memory is important when determining processability. If multiple phases are processed concurrently, many buffers are used, which can cause out-of-memory (OOM) problems. Moreover, if many data transfers occur at the same time, bottlenecks could occur during data transfers due to limited memory bandwidth, which could slow down processing.

In particular, the proposed method is compatible not only with a single DLA/GPU platform but also with various hardware platforms (*e.g.*, multiple DLA and GPU platforms) in the same way. For example, in the case of two DLAs and two GPUs, four processing phases can be processed simultaneously by assigning them to each of the DLAs and GPUs by 4-partitioning as shown in Fig. 5, and consequently, this can improve the processing speed by increasing the hiding processing time. Therefore, the proposed method is easily compatible with various hardware platforms regardless of the number of cores in the processing unit. Even considering the multi-camera environment, since the perception of the camera input is processed sequentially in units of frames and the planning and control processes are performed after perception by combining the perception results, there is no dependency between each frame input from multiple cameras [58]. Therefore, the proposed method can be practically applied to a multi-camera platform with multiple 2-partitioning structures. Similarly, even considering a

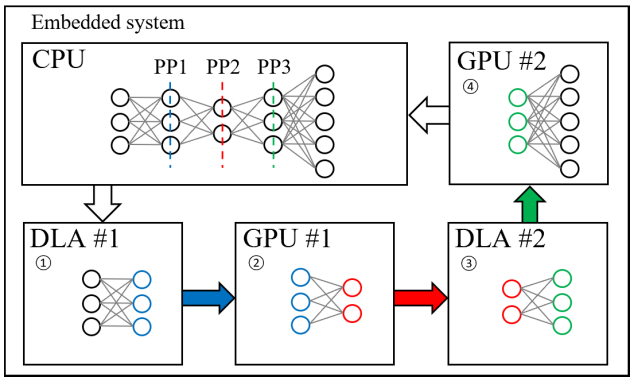

FIGURE 5. 4-partition process on the platform with two DLAs and two GPUs.

multi-sensor system (*e.g.*, camera + LiDAR + RADAR), the proposed method processes the independent inputs of each sensor in parallel. Therefore, it can be universally applied to the input processing of all sensors to achieve latency and power reduction effects. Consequently, considering the various problems mentioned here, partitioning the network to minimize the idle/data transfer time and processing the network in parallel by DLA and GPU would result in the network inference being optimized for a given system. In addition, it is noteworthy that the proposed network inference with the optimal PP does not incur additional costs, such as retraining, because the proposed method uses pre-trained weights, as shown in Fig. 4. This differs significantly from many recently proposed lightweight techniques that involve retraining requiring considerable time and resources [59], [60].

C. APPLICATION TO DEEP CONVOLUTIONAL NEURAL NETWORKS

The application process of the proposed method can be summarized as follows: (1) to identify the network architecture, (2) to search the optimal PP of the network with Algorithm1, and (3) to apply network pipelining of the DLA and GPU based on the optimal PP. This subsection presents the process of applying the proposed method to various CNN-based object detectors with excellent compatibility by focusing on a 2-partition structure. First, it is important to understand the network structure including the dependencies of each layer. In general, CNNs use a backbone network that stacks convolution layers for feature extraction on the front part and use a custom layer suitable for a specific task (*e.g.*, detection, segmentation) on the back part. It is important to note that the backbone networks can be completely processed in DLA, but the custom layers for a specific task in the back part cannot be processed in DLA. For example, ResNet [42], a representative classification network widely used as the backbone of many other networks, is composed of convolution, pooling, and fully connected layers; thus, it can be fully processed in DLA. However, in the case of YOLACT [61], a representative semantic segmentation network, the ResNet (*i.e.*, backbone) and FPN [62] layers located in the front part can be supported by DLA, but the layer that predicts the convolution output in the back part cannot be supported by DLA. Similarly, YOLOv3 [36], a representative object detector, also contains a yolo layer on the back part, which is a custom layer not supported by DLA. Therefore, if DLA processes these custom layer operations in the back part, calculations that DLA cannot handle must be transferred to GPU for processing, which would cause unnecessary data transfer latency. Consequently, it is efficient to design DLA to process the front part of the CNNs to prevent unnecessary data transfer. It is noteworthy that most CNN-based object detectors for autonomous driving generally use backbones for feature extraction, and most backbone networks consist of convolutional layers [63]. Therefore, even if DLA processes only a part of the convolution layers through the proposed method, the latency and power performance can be improved in most CNN-based object detectors.

Second, the calculation of the optimal network PP is possible by analyzing the device performance of DLA and GPU in the embedded platform. Let T_{dla} and T_{gpu} be the ratio of tasks divided to the DLA and GPU based on the PP, respectively, and CP_{gpu} and CP_{dla} denote GPU computing power and DLA computing power, respectively. Subsequently, the following equations are established:

$$T_{dla} + T_{gpu} = 1 \tag{1}$$

$$\frac{T_{dla}}{CP_{dla}} = \frac{T_{gpu}}{CP_{gpu}} \tag{2}$$

Based on these equations, T_{dla} and T_{gpu} should be determined to satisfy (2) because the throughputs of the two devices should be the same for an ideal parallel structure. From (1) and (2), the optimal split ratio of tasks allocated to DLA is calculated as follows:

$$T_{dla} = \frac{CP_{dla}}{CP_{dla} + CP_{gpu}} \tag{3}$$

For example, if the computing power of GPU in FLOPS is twice that of DLA, then $CP_{gpu} = 2 \times CP_{dla}$ is satisfied. Therefore, one-third of the entire network should be allocated to DLA to fully utilize the two devices without incurring idle time.

Finally, as mentioned in Section III-B, the data transfer time between DLA and GPU should be considered when selecting a network PP. For example, if there is a layer with dependencies in addition to the partitioned output layer, the data to be transmitted increases. Therefore, selecting a layer with a short data transmission time that has no additional dependencies as the optimal PP is also important. In addition, the data transfer time affects the actual processing speed, even if there is no dependency, and is proportional to the output feature map size of the layer before the PP. In general, the feature map size decreases in the back part of the layers. Therefore, the final optimal PP, including the data transfer time, is selected by comparing the PP selected according to the device computing power and amount of computation with the following PPs (*i.e.*, layers) in the next position.

Algorithm 1 The Network Partitioning Point Search Algorithm

Input: N, O_i, t_{dla_i}, CP_{device}
N: *Total number of layers*
O_i: *The number of operations 0 to* i^{th} *layer*
t_{dla_i}: *Sum of operation time of DLA and data transfer time with* i^{th} *PP*
CP_{device}: *Computing power of device*
Output: PP
PP: Partitioning point of network

```
i ← 0
while | Oi/ON | ≤ | CPdla/(CPgpu+CPdla) | do
    PP ← i
    while t_dla_PP > t_dla_PP+1 do
        PP ← PP + 1
    end while
    i ← i + 1
end while
```

Algorithm1 describes a method to search for the optimal PP that minimizes data transfer time (*i.e.*, 4*th*, 5*th* lines), in addition to the computing power of a device and the amount of computation based on FLOPs (*i.e.*, 2*nd*, 3*rd* lines). As a result, the proposed method has high compatibility because only a simple network analysis, device performance analysis, and data transfer time analysis are required.

D. SUMMARY OF STRENGTHS OF THE PROPOSED METHOD

The advantages of the proposed method are summarized as follows:

- First, the most important advantage is that there is no loss of accuracy because the target application fully utilizes the hardware resources of the embedded system without changing the network structure. It should be noted that in real autonomous vehicles, accuracy is paramount to ensure safety.
- Second, there is no additional cost such as re-training because the method uses the pre-trained weights as they are. This differs greatly from the fact that many of the recently proposed lightweight techniques [25], [64] basically involve re-training, and consequently, the proposed method saves time and resources that would have been used for re-training.
- Third, the proposed method is advantageous in terms of power efficiency as well as processing speed because operations that were entirely performed on GPU (with its poor power efficiency) are effectively divided and computed on both DLA (which is more efficient with respect to power consumption) and GPU.
- Lastly, the proposed method has the advantage of being highly scalable and can be applied to various CNN-based object detectors in heterogeneous system environments. Considering the characteristics of deep learning algorithms, where new superior networks are rapidly being proposed, this advantage is significant in that the method can continue supporting the algorithm even if the algorithm is constantly updated.

IV. EXPERIMENTAL RESULTS

A. EXPERIMENTAL ENVIRONMENTS

The superiority of the proposed technique is demonstrated by conducting experiments with various object detection algorithms on a Jetson Xavier AGX board, an embedded platform for autonomous driving. The Xavier board is a heterogeneous embedded system equipped with a Volta GPU (512 CUDA cores) and nvDLA developed by NVDIA. The server environment used for the comparison contains a Pascal 1080ti GPU (3584 CUDA cores). The Xavier board provides seven modes according to the clock frequencies of each processor and memory controller [65]. In this study, mode 0, which uses the maximum clock frequency for high performance, is used to fully utilize the hardware resources because rapid and accurate operation processing is most important in the autonomous driving environment we assumed. Since the nvDLA can be controlled only through TensorRT [48], in this experiment, the networks applying the proposed method are implemented and used as tensorRT. The software environment comprises CUDA 10.0, cuDNN version 7, and TensorRT version 5.1.6.1 [48]. For accurate performance evaluation as a solution for autonomous driving, the validation set of the Berkeley Deep Drive (BDD) [66] dataset, a representative autonomous driving dataset, is used for accuracy comparison, and the execution time and energy efficiency are evaluated on the Jetson Xavier AGX board. The average power consumption of the server GPU system and Jetson Xavier AGX embedded board is measured using the cuda nvidia-smi API [67] and the tegrastats utility included in the NVIDIA JetPack SDK [68], respectively. All codes and real-time demo videos related to this study are available at: https://github.com/jjeonda/CP-CNN

B. APPLICATION OF THE PROPOSED METHOD TO OBJECT DETECTORS ON THE JETSON XAVIER AGX

In this subsection, we present the practical application of the proposed method to object detectors on a Jetson Xavier AGX board. First of all, the process of applying the proposed method to YOLOv3 [36], a representative object detector, is as follows. As shown in Fig. 6(a), YOLOv3 is composed of a residual block with shortcut, route, and yolo layers. The shortcut layer is a skip connection [42], and the route layer returns concatenated feature maps of the indexed layers. The yolo layer is a detection layer used in the YOLOv3 network; consequently, the back part, including the yolo layer, should be processed on GPU. Next, the performance of the two devices in the heterogeneous embedded platform (*i.e.*, DLA and GPU) should be considered for optimal parallelization. In the Jetson Xavier AGX, the computational performance of GPU and DLA is 11 TFLOPS and 2.5 TFLOPS, respectively, based on the FP16 format [65]. Thus, the performances of

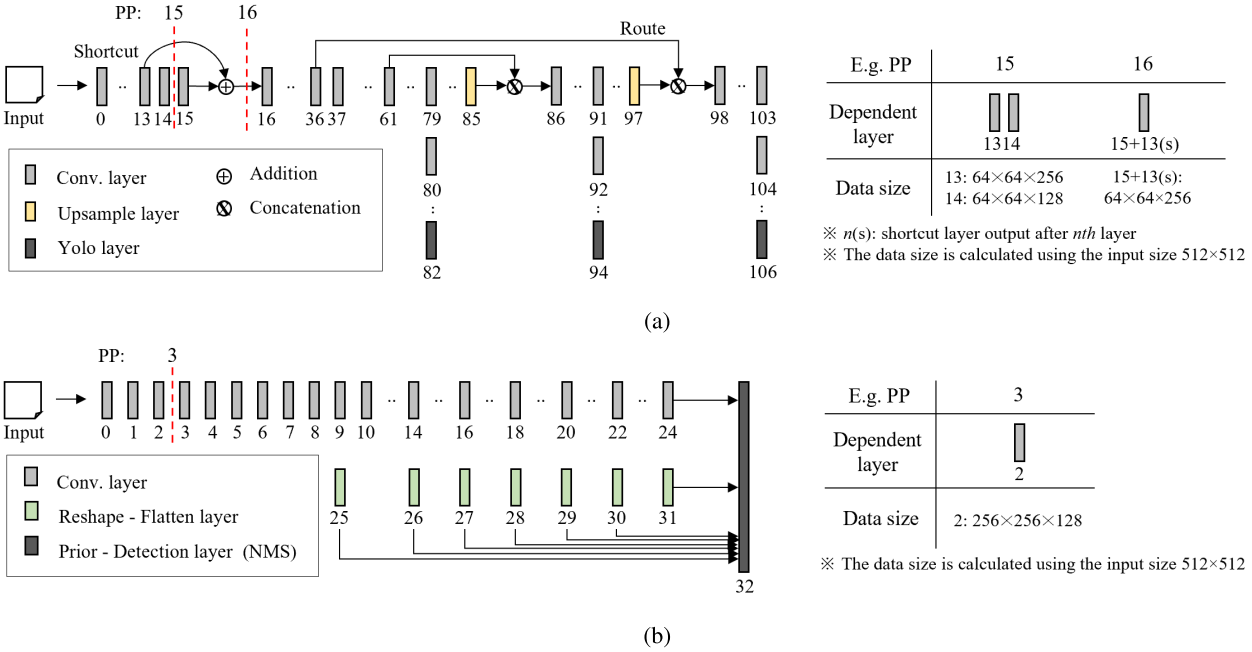

FIGURE 6. Network structure by layers considering the layer dependency and DLA support availability: (a) YOLOv3, (b) SSD. The operation sequence is expressed by the layer number and arrow direction. Even if the proposed method is applied, the operation sequence is the same except that the processing device is different according to the PP.

the two devices differ by approximately 4.4× based on the FLOPS; thereby, a layer corresponding to 18.52% of the total computational amount is selected as the PP. Finally, the data transfer time between DLA and GPU should be considered, and the transmission of the feature size 64 × 64 × 256 from DLA to GPU within the experimental environments requires approximately 2 ms of data transfer time. This time is approximately 3.88% of GPU processing speed based on the TensorRT API [48] YOLOv3; hence, many data transfers between the devices degrade the performance of the application. In YOLOv3 depicted in Fig. 6(a), when setting the 15*th* layer as the PP and processing the preceding part with DLA and the subsequent part with GPU, the output of the 13*th* layer (*i.e.*, the shortcut layer), as well as that of the 14*th* layer, must be copied to GPU owing to dependencies. Therefore, according to the 4*th* and 5*th* lines of Algorithm1, 16*th* layers should be used as the PP.

TABLE 1. Execution time and energy efficiency according to the partitioning points.

Network	Partitioning Point	Time(ms)	Power(W)	Energy/Image(mJ)
YOLOv3	9 (10.76%)	14.63	24.09	352.46
	16 (18.62%)	**12.96**	**23.66**	**306.61**
	22 (24.04%)	14.19	22.66	321.55
	28 (29.47%)	15.67	21.18	331.86
	34 (34.89%)	17.50	19.23	336.54
SSD	4 (26.28%)	68.22	38.26	2610.37
	5 (31.44%)	**32.94**	**27.11**	**893.07**
	6 (41.76%)	40.46	23.50	950.85
	7 (52.08%)	47.81	38.70	1850.27

Table 1 lists the processing time per frame and the power consumption in FP16 operation according to the PPs when the network inference with an image input size of 512 × 512 is processed by applying the proposed method to the YOLOv3 network. We retrain the YOLOv3 network with the ReLU activation function because nvDLA does not support leaky ReLU. The second column indicates the network PP on each network, and the numbers in parentheses indicate the percentage of operations before the network PP in the entire network. As mentioned above, the hardware utilization of DLA and GPU can be optimized when using a layer that is 18.52% of the total network computation as a PP, and the 16*th* layer is determined to be the optimal PP according to Algorithm1. It should be noted that the number of floating operations up to the 16*th* layer is 18.62% of the total number of convolutional operations, which closely approximates 18.52%. In other words, because the computing power ratio (*i.e.*, 11TFLOPs:2.5TFLOPs) and the measured hardware utilization ratio (*i.e.*, 6.22GFLOPs:1.42GFLOPs) of GPU and DLA are similar, it is reasonable to use the computing power of each device for the estimation of execution time.

Fig. 7 shows the actual profiling results [69] of YOLOv3 on the embedded board according to various PPs (*i.e.*, 9, 16, and 34) to check whether the devices are fully operated without the idle time. Fig. 7(a) shows the profiling result when using the optimal PP (*i.e.*, 16) obtained by Algorithm1. Since idle time does not occur in DLA and GPU during operations, the processing time for one phase is the shortest. On the other hand, in Fig. 7(b), as the number of layers processed by the GPU increases, the one phase processing time becomes longer compared to the optimal case in Fig. 7(a). This indicates that DLA idle time longer than the increased GPU

TABLE 2. Performance evaluation of various networks on various platforms.

System	Precision	Device	YOLOv3			SSD			YOLOv5s		
			Time (ms)	Energy/Image (mJ)	mAP (%)	Time (ms)	Energy/Image (mJ)	mAP (%)	Time (ms)	Energy/Image (mJ)	mAP (%)
Server	FP32	GPU	22.64	5338.85	14.21	43.3	4599.4	14.10	7.98	-	18.70
Embedded	FP32	GPU	58.40	1914.24	14.21	134.6	3700.9	14.10	-	-	-
		GPU-TRT	51.42	1589.31	14.21	114.8	3957.4	14.10	11.68	354.67	18.70
Embedded	FP16	GPU-TRT	18.10	329.24	14.21	46.3	969.4	14.10	8.84	153.53	18.70
		DLA-TRT	38.95	371.19	14.21	144.6	1565.6	14.10	17.70	178.45	18.70
		Proposed-TRT (DLA&GPU)	**12.96**	**306.61**	**14.21**	**32.9**	**893.1**	**14.10**	**5.23**	**133.88**	**18.70**

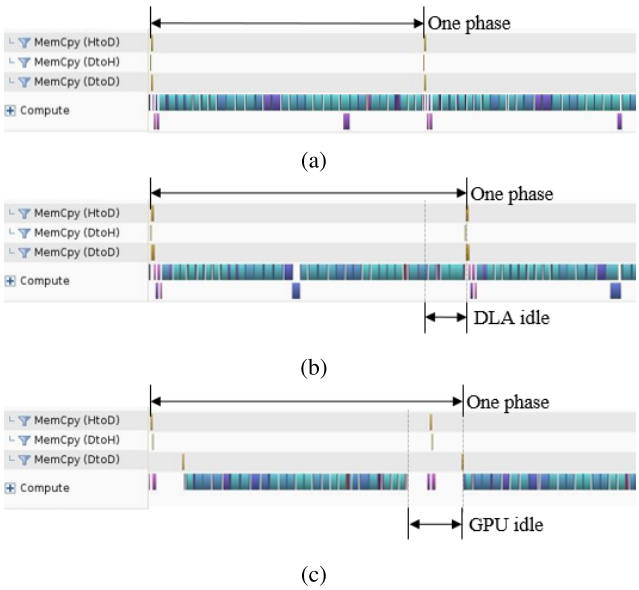

FIGURE 7. YOLOv3 profiling results according to the partitioning points: (a) 16, (b) 9, and (c) 34.

processing time occurs. In Fig. 7(c), the DLA operation time is longer than the GPU operation time because the number of layers processed by the GPU is smaller than in Fig. 7(a). As a result, there is a GPU idle time until the next phase DLA calculation result is received (*i.e.*, MemCpy(DtoD)), which leads to a longer one-phase processing time. These profiling results show that the timing diagrams in Fig. 3 are reflected in the embedded board, showing that assigning the optimal network PP ensures the optimized operating speed even on real embedded boards.

The scalability and compatibility of the proposed method can be demonstrated by applying it to SSD [37] using the same approach as YOLOv3. As shown in Fig. 6(b), SSD uses a multi-scale feature map architecture that includes priorbox, reshape, flatten, and detection layers in addition to convolution layers. The priorbox layer performs bounding box prediction, and the reshape layer changes the dimension. The flatten layer performs softmax, and the detection layer calculates non-maximum suppression (NMS). Because the layers included in the back part of the SSD are also not supported by DLA, the back part should be processed by GPU. In addition, considering the computational performance of DLA and GPU, 18.52% of the total computational amount should be selected as the PP, and the data transfer time should be minimized using Algorithm1. In detail, first of all, the 3*rd* layer is determined as a PP according to the 2*nd* and 3*rd* lines of Algorithm1. Subsequently, based on the consideration of the data transfer time in the 4*th* and 5*th* lines of Algorithm1, the optimal PP is determined as the 5*th* layer. This is because SSD performs NMS calculations that affect the data transfer time as many times as the number of boxes in the feature extraction layer, resulting in a speed bottleneck. Consequently, as shown in the four rows in the lower part of Table 1, the processing time and energy efficiency are the best when the SSD is divided at the 5*th* layer. It is noteworthy that SSD has a lower performance improvement than YOLOv3 due to the gap between the ratio of the computing power of devices and the optimal PP determined by Algorithm1. However, the energy efficiency and processing speed of the proposed method are still outstanding compared to the baseline algorithm, making it suitable for operation in an embedded platform environment.

C. PERFORMANCE EVALUATION

Table 2 lists the processing time per frame, energy per image, and mean average precision (mAP) of YOLOv3, SSD, and YOLOv5 small model (YOLOv5s) [38], during network inference with an image input size of 512 × 512 on various computing platforms. The optimal PPs of YOLOv3 and SSD follow the results in Table 1, and when the proposed method is applied to YOLOv5s, conv3, which is the 16.52% of the total number of convolutional operations, is determined as optimal PP according to the proposed method. The experimental results show that the server environment consumes much more power than the embedded environment, but the processing speed of the server GPU environment (3*rd* row) is higher than that of the embedded environment (4*th* row). This implies that the processing speed of an algorithm capable of real-time operation in a server environment is degraded when used as it is on an embedded board with limited resources. This constitutes a limitation for autonomous driving applications where real-time processing is essential. Accelerating the same model using only GPU with the TRT library (GPU-TRT) without retraining (5*th* row) enables faster processing times than the baseline embedded GPU system (4*th* row). In addition, as shown in the comparison between 5*th* row and 6*th* row, FP16 achieves

much faster processing time and superior energy efficiency than FP32 on the same platform. It should be noted that there is no difference in the accuracy between FP16 and FP32 [70]. When using only DLA with the TRT library (*i.e.*, DLA-TRT in the 7*th* row), the power consumption per second is approximately twice as low as that when using only GPU with the TRT library (*i.e.*, GPU-TRT in the 6*th* row), but the time taken to process one phase is more than twice as long; thus, the energy efficiency (*i.e.*, energy per image) is worse. Finally, when the proposed parallelization method with the optimal PP is applied (*i.e.*, Proposed-TRT in the 8*th* row), the processing speed and the energy efficiency are enhanced compared with the best-performing GPU-TRT with the FP16 platform among existing platforms. It is noteworthy that the mAP of each model is the same on all platforms as none of the platforms mentioned above incurs a loss of accuracy. In detail, the proposed method enhances the processing speed by 45.44 (77.8%) and 101.7 (75%) ms and the power efficiency by 1.61 (84%) and 2.81 (75.9%) J/image on YOLOv3 and SSD, respectively, compared with the baseline embedded GPU system (*i.e.*, FP32 on the GPU platform). In addition, compared to the GPU-TRT with the FP16 platform, the proposed method on YOLOv3, SSD, and YOLOv5s not only improves the processing speed by 5.14ms (28.4%), 13.4ms (28.9%), and 3.61ms (40.8%), but also enhances the power efficiency by 22.63 mJ/image (6.9%), 76.3 mJ/image (7.9%), and 19.65 mJ/image (12.8%), respectively. The results of these experiments clearly show that the proposed method additionally utilizes energy-efficient DLA together with GPU in a heterogeneous system to increase the processing speed and energy efficiency without loss of accuracy. As a result, the proposed method enables real-time processing by overcoming performance degradation when implementing algorithms in embedded systems with limited resources. In addition, all these experimental results show that the proposed method can be generally applied to various CNN-based object detectors due to its high scalability and compatibility.

TABLE 3. Performance comparison of execution time, energy efficiency, and mAP.

Network	Time(ms)	Energy/Image(mJ)	mAP(%)
Joint-OP-Tiny-YOLOv2 [29]	18.10	531.86	4.48
Joint-OP-Tiny-YOLOv3 [29]	24.85	680.30	6.60
YOLOv3 [36]	58.40	1914.24	14.21
Proposed-YOLOv3	**12.96**	**306.61**	**14.21**
Tiny-SSD [46]	26.8	323.6	5.56
MobileNet-SSD [30]	18.0	302.4	6.61
SSD [37]	134.6	3700.9	14.10
Proposed-SSD	**32.9**	**893.1**	**14.10**

D. PERFORMANCE COMPARISON

To demonstrate the superiority of the proposed method, the performance of the proposed scheme is compared with that of existing networks [29], [30], [36], [37], [46] in an embedded system environment. Table 3 presents the processing time per frame, energy per image, and mAP of the proposed method and existing networks when the input frame size is 512×512. The joint optimization of Tiny-YOLO [29], Tiny-SSD [46], and MobileNet-SSD [30], which improves the processing speed and power efficiency by optimizing each network, has higher processing speed and power efficiency compared with the baseline algorithm. However, the accuracy degradation is problematic because it is unacceptably severe. Conversely, the proposed parallel processing on DLA and GPU enhances the processing speed and power efficiency and yields the best mAP. Consequently, the proposed method is the most suitable for operating a real-time low-power deep learning algorithm based on a heterogeneous embedded platform for autonomous driving.

V. CONCLUSION

In this paper, we propose a method to optimize hardware utilization through parallel processing of GPU and DLA in heterogeneous embedded systems without changing the model structure (*i.e.*, without loss of accuracy). In particular, the GPU and power-efficient DLA are fully utilized without idle time to enhance processing speed and power efficiency. Another advantage of the proposed method is that it does not require additional costs such as re-training or lightweight techniques, and can be easily applied to various CNN-based object detectors due to its high compatibility. The application of the proposed method to YOLOv3 and SSD, which are representative object detectors, improves the processing speed by 77.8% and 75.6%, respectively, and the energy efficiency by 84% and 75.9%, respectively, compared with the baseline system. As a result, the proposed method is the most suitable for autonomous driving applications because it achieves the best trade-off between accuracy, processing speed, and power efficiency, and has excellent scalability. In conclusion, this paper proposes a standalone solution for camera input processing that requires the most power and latency in multi-sensor systems and can be compatible with a multi-autonomous driving system. Moreover, it is noteworthy that the proposed method has the advantage of being easily extended to various embedded platform-based image processing applications.

REFERENCES

[1] J. Choi, D. Chun, H. Kim, and H.-J. Lee, "Gaussian YOLOv3: An accurate and fast object detector using localization uncertainty for autonomous driving," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 502–511.

[2] S. Pang, D. Morris, and H. Radha, "CLOCs: Camera-LiDAR object candidates fusion for 3D object detection," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 10386–10393.

[3] R. Nabati and H. Qi, "CenterFusion: Center-based radar and camera fusion for 3D object detection," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2021, pp. 1526–1535.

[4] Y. Wu, Y. Wang, S. Zhang, and H. Ogai, "Deep 3D object detection networks using LiDAR data: A review," *IEEE Sensors J.*, vol. 21, no. 2, pp. 1152–1171, Jan. 2021.

[5] J. Choi, D. Chun, H.-J. Lee, and H. Kim, "Uncertainty-based object detector for autonomous driving embedded platforms," in *Proc. 2nd IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Aug. 2020, pp. 16–20.

[6] *NVIDIA Deep Learning Accelerator*, NVIDIA, Santa Clara, CA, USA, 2018.
[7] *The Evolution of EyeQ*, MOBILEYE, Jerusalem, Israel, 2020.
[8] Y. Li, A. Dua, and F. Ren, "Light-weight RetinaNet for object detection on edge devices," in *Proc. IEEE 6th World Forum Internet Things (WF-IoT)*, Jun. 2020, pp. 1–6.
[9] M. Xia, Z. Huang, L. Tian, H. Wang, V. Chang, Y. Zhu, and S. Feng, "SparkNoC: An energy-efficiency FPGA-based accelerator using optimized lightweight CNN for edge computing," *J. Syst. Archit.*, vol. 115, May 2021, Art. no. 101991.
[10] S. P. Kaarmukilan, S. Poddar, and K. A. Thomas, "FPGA based deep learning models for object detection and recognition comparison of object detection comparison of object detection models using FPGA," in *Proc. 4th Int. Conf. Comput. Methodol. Commun. (ICCMC)*, Mar. 2020, pp. 471–474.
[11] J. Tang, S. Liu, L. Liu, B. Yu, and W. Shi, "LoPECS: A low-power edge computing system for real-time autonomous driving services," *IEEE Access*, vol. 8, pp. 30467–30479, 2020.
[12] D. T. Nguyen, T. N. Nguyen, H. Kim, and H. Lee, "A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1861–1873, Aug. 2019.
[13] D. T. Nguyen, H. Kim, H.-J. Lee, and I.-K. Chang, "An approximate memory architecture for a reduction of refresh power consumption in deep learning applications," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–5.
[14] D. T. Nguyen, H. Kim, and H.-J. Lee, "Layer-specific optimization for mixed data flow with mixed precision in FPGA design for CNN-based object detectors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 6, pp. 2450–2464, Jun. 2021.
[15] H. Mo, L. Liu, W. Zhu, Q. Li, S. Yin, and S. Wei, "A 460 GOPS/W improved mnemonic descent method-based hardwired accelerator for face alignment," *IEEE Trans. Multimedia*, vol. 23, pp. 1122–1135, 2021.
[16] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. 14th Annu. Int. Conf. Mobile Syst., Appl., Services*, Jun. 2016, pp. 123–136.
[17] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "DeepX: A software accelerator for low-power deep learning inference on mobile devices," in *Proc. 15th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2016, pp. 1–12.
[18] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, "μLayer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization," in *Proc. 14th EuroSys Conf.*, Mar. 2019, pp. 1–15.
[19] A. Zlateski, K. Lee, and H. S. Seung, "ZNNi: Maximizing the inference throughput of 3D convolutional networks on CPUs and GPUs," in *Proc. Int. Conf. for High Perform. Comput., Netw., Storage Anal. (SC)*, Nov. 2016, pp. 854–865.
[20] S. Hossain and D.-J. Lee, "Deep learning-based real-time multiple-object detection and tracking from aerial imagery via a flying robot with GPU-based embedded devices," *Sensors*, vol. 19, no. 15, p. 3371, Jul. 2019.
[21] Y. Cai, H. Li, G. Yuan, W. Niu, Y. Li, X. Tang, B. Ren, and Y. Wang, "YOLObile: Real-time object detection on mobile devices via compression-compilation co-design," in *Proc. AAAI Conf. Artif. Intell.*, vol. 35, 2021, pp. 955–963.
[22] Y. Xu, H. Wu, W. Zhang, C. Yang, Y. Wu, H. Gao, and T. Wang, "Talos: A weighted speedup-aware device placement of deep learning models," in *Proc. IEEE 32nd Int. Conf. Appl.-Specific Syst., Architectures Processors (ASAP)*, Jul. 2021, pp. 101–108.
[23] W. Zhang, H. Sun, D. Zhao, L. Xu, X. Liu, H. Ning, J. Zhou, Y. Guo, and S. Yang, "A streaming cloud platform for real-time video processing on embedded devices," *IEEE Trans. Cloud Comput.*, vol. 9, no. 3, pp. 868–880, Jul. 2021.
[24] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.
[25] S. Kim and H. Kim, "Zero-centered fixed-point quantization with iterative retraining for deep convolutional neural network-based object detectors," *IEEE Access*, vol. 9, pp. 20828–20839, 2021.
[26] N. J. Kim and H. Kim, "FP-AGL: Filter pruning with adaptive gradient learning for accelerating deep convolutional neural networks," *IEEE Trans. Multimedia*, early access, Jul. 11, 2022, doi: 10.1109/TMM.2022.3189496.
[27] Z. Wang, W. Hong, Y.-P. Tan, and J. Yuan, "Pruning 3D filters for accelerating 3D ConvNets," *IEEE Trans. Multimedia*, vol. 22, no. 8, pp. 2126–2137, Aug. 2020.
[28] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "ENet: A deep neural network architecture for real-time semantic segmentation," 2016, *arXiv:1606.02147*.
[29] D. Kang, D. Kang, J. Kang, S. Yoo, and S. Ha, "Joint optimization of speed, accuracy, and energy for embedded image recognition systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 715–720.
[30] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
[31] S. Mittal, "A survey on optimized implementation of deep learning models on the NVIDIA Jetson platform," *J. Syst. Archit.*, vol. 97, pp. 428–442, Aug. 2019.
[32] H. Guo, H. Bai, Y. Zhou, and W. Li, "DF-SSD: A deep convolutional neural network-based embedded lightweight object detection framework for remote sensing imagery," *Proc. SPIE*, vol. 14, no. 1, p. 014521, 2020.
[33] Q. Zhou, J. Wang, J. Liu, S. Li, W. Ou, and X. Jin, "RSANet: Towards real-time object detection with residual semantic-guided attention feature pyramid network," *Mobile Netw. Appl.*, vol. 26, no. 1, pp. 77–87, Feb. 2021.
[34] B. Wu, A. Wan, F. Iandola, P. H. Jin, and K. Keutzer, "SqueezeDet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 446–454.
[35] P. Li, H. Zhao, P. Liu, and F. Cao, "RTM3D: Real-time monocular 3D detection from object keypoints for autonomous driving," in *Proc. Eur. Conf. Comput. Vis.* Singapore: Springer, 2020, pp. 644–660.
[36] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.
[37] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.* Singapore: Springer, 2016, pp. 21–37.
[38] G. Jocher et al., "Ultralytics/YOLOv5: V7.0-YOLOv5 SOTA real-time instance segmentation," Zenodo, Tech. Rep., 2022, doi: 10.5281/zenodo.7347926.
[39] Y. Zhou, A. Mao, S. Huo, J. Lei, and S.-Y. Kung, "Salient object detection via fuzzy theory and object-level enhancement," *IEEE Trans. Multimedia*, vol. 21, no. 1, pp. 74–85, Jan. 2019.
[40] J. Li, X. Liang, J. Li, Y. Wei, T. Xu, J. Feng, and S. Yan, "Multistage object detection with group recursive learning," *IEEE Trans. Multimedia*, vol. 20, no. 7, pp. 1645–1655, Jul. 2018.
[41] S. Oh, J.-H. You, and Y.-K. Kim, "FRDet: Balanced and lightweight object detector based on fire-residual modules for embedded processor of autonomous driving," 2020, *arXiv:2011.08061*.
[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[43] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50× fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*.
[44] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6517–6525.
[45] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," 2015, *arXiv:1511.06530*.
[46] A. Womg, M. J. Shafiee, F. Li, and B. Chwyl, "Tiny SSD: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection," in *Proc. 15th Conf. Comput. Robot Vis. (CRV)*, May 2018, pp. 95–101.
[47] D. Zhang, M. Cui, Y. Yang, P. Yang, C. Xie, D. Liu, B. Yu, and Z. Chen, "Knowledge graph-based image classification refinement," *IEEE Access*, vol. 7, pp. 57678–57690, 2019.
[48] *Tensorrt Developer's Guide*, NVIDIA Corp., Santa Clara, CA, USA, 2020.
[49] J. Lee, J. Jang, J. Lee, D. Chun, and H. Kim, "CNN-based mask-pose fusion for detecting specific persons on heterogeneous embedded systems," *IEEE Access*, vol. 9, pp. 120358–120366, 2021.
[50] Y. Xu, W. Dai, Y. Qi, J. Zou, and H. Xiong, "Iterative deep neural network quantization with Lipschitz constraint," *IEEE Trans. Multimedia*, vol. 22, no. 7, pp. 1874–1888, Jul. 2020.

[51] J. M. Rodriguez-Borbon, X. Ma, A. K. Roy-Chowdhury, and W. A. Najjar, "Heterogeneous acceleration of HAR applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 3, pp. 888–902, Mar. 2020.
[52] J. Yang, Y. Yang, Z. Chen, L. Liu, J. Liu, and N. Wu, "A heterogeneous parallel processor for high-speed vision chip," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 3, pp. 746–758, Mar. 2018.
[53] X. Ding, T. Hao, J. Tan, J. Liu, J. Han, Y. Guo, and G. Ding, "ResRep: Lossless CNN pruning via decoupling remembering and forgetting," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 4490–4500.
[54] P. Chen, J. Liu, B. Zhuang, M. Tan, and C. Shen, "AQD: Towards accurate quantized object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 104–113.
[55] O. Spantidi, G. Zervakis, I. Anagnostopoulos, H. Amrouch, and J. Henkel, "Positive/negative approximate multipliers for DNN accelerators," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2021, pp. 1–9.
[56] M. Yang, N. Otterness, T. Amert, J. Bakita, J. H. Anderson, and F. D. Smith, "Avoiding pitfalls when using NVIDIA GPUS for real-time tasks in autonomous systems," in *Proc. 30th Euromicro Conf. Real-Time Syst. (ECRTS)*. Wadern, Germany: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018, pp. 1–21.
[57] J. Luitjens, "CUDA streams: Best practices and common pitfalls," in *Proc. GPU Techonology Conf.*, 2015, pp. 1–71.
[58] Z. Li, W. Wang, H. Li, E. Xie, C. Sima, T. Lu, Q. Yu, and J. Dai, "BEVFormer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers," 2022, *arXiv:2203.17270*.
[59] Y. He, Y. Ding, P. Liu, L. Zhu, H. Zhang, and Y. Yang, "Learning filter pruning criteria for deep convolutional neural networks acceleration," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2006–2015.
[60] R. Yu, A. Li, C. Chen, J. Lai, V. I. Morariu, X. Han, M. Gao, C. Lin, and L. S. Davis, "NISP: Pruning networks using neuron importance score propagation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9194–9203.
[61] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "YOLACT: Real-time instance segmentation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9156–9165.
[62] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 936–944.
[63] Q. Zhao, T. Sheng, Y. Wang, Z. Tang, Y. Chen, L. Cai, and H. Ling, "M2Det: A single-shot object detector based on multi-level feature pyramid network," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 9259–9266.
[64] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 11256–11264.
[65] D. Franklin, "Jetson AGX Xavier and the new era of autonomous machines webinar," NVIDIA, Tech. Rep., 2020. [Online]. Available: https://info.nvidia.com/jetsonxavier-and-the-new-era-of-autonomous-machines-reg-page.html
[66] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "BDD100K: A diverse driving dataset for heterogeneous multitask learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2633–2642.
[67] *NVIDIA System Management Interface*, NVIDIA, Santa Clara, CA, USA, 2016.
[68] *NVIDIA Jetson Linux Driver Package Software Features*, NVIDIA, Santa Clara, CA, USA, 2019.
[69] *Profiler User's Guide*, NVIDIA, Santa Clara, CA, USA, 2020.
[70] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.

DAYOUNG CHUN (Graduate Student Member, IEEE) received the B.S. degree in electronics engineering from Sogang University, Seoul, South Korea, in 2018. She is currently pursuing the integrated M.S. and Ph.D. degree in electrical and computer engineering with Seoul National University, Seoul. Her research interests include the algorithms and architectures of deep learning and GPU architectures for computer vision.

JIWOONG CHOI received the B.S. degree in electrical and electronics engineering from Chung-Ang University, Seoul, South Korea, in 2015, and the M.S. and Ph.D. degrees in electrical and computer engineering from Seoul National University, Seoul, in 2017 and 2021, respectively. He is currently a Deep Learning Research Engineer with NVIDIA, Santa Clara, CA, USA.

HYUK-JAE LEE (Member, IEEE) received the B.S. and M.S. degrees in electronics engineering from Seoul National University, South Korea, in 1987 and 1989, respectively, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 1996. From 1998 to 2001, he was with the Server and Workstation Chipset Division, Intel Corporation, Hillsboro, OR, USA, as a Senior Component Design Engineer. From 1996 to 1998, he was with the Faculty of the Department of Computer Science, Louisiana Tech University, Ruston, LS, USA. In 2001, he joined the School of Electrical Engineering and Computer Science, Seoul National University, where he is currently a Professor. He is also the Founder with Mamurian Design Inc., where he is focusing on the fabless SoC design house for multimedia applications. His research interests include computer architecture and SoC design for multimedia applications.

HYUN KIM (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2009, 2011, and 2015, respectively. From 2015 to 2018, he was with the BK21 Creative Research Engineer Development for IT, Seoul National University, Seoul, as a BK Assistant Professor. In 2018, he joined the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, where he is currently an Associate Professor. His research interests include algorithm, computer architecture, memory, and SoC design for low-complexity multimedia applications and deep neural networks.