Journal of KIIT. Vol. 22, No. 9, pp. 89-100, Sep. 30, 2024. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2024.22.9.89

CBL 블록 기반 피라미드 형태의 다중계층 슬라이딩 윈도우를 위한 하드웨어 구조

김경욱*, 문병인**

A Hardware Architecture for the CBL Block based Pyramid-shaped Multilayer Sliding Window

Kyeonguk Kim*, Byungin Moon**

요 약

최근 실시간 임베디드 플랫폼에서 YOLOv4-tiny는 빠른 처리속도와 높은 정확성으로 주목받고 있으나, 시스톨릭 어레이 구조의 한계로 실시간 환경에 적용하는 것은 한계가 있었다. 이를 해결하기 위해 Fused Layer 구조가 제안되었지만 YOLOv4-tiny의 CBL(Convolution Operation, Batch Normalization, Leaky ReLU) 블록에 최적화되지 않아 여전히 실시간 처리속도의 성능 보장에 어려움이 있다. 이에 본 논문에서는 CBL 블록에 적합한 Fused Layer 구조를 제안한다. 제안하는 구조는 시높시스의 14nm 오픈 라이브러리를 통해 합성되었으며, Systolic Array 구조 대비 처리량이 296.74% 향상되고, latency가 74.76% 감소하는 것을 확인하였다. 그러므로 제안하는 구조는 YOLOv4-tiny의 실시간 처리속도 보장에 기여할 수 있으며, 최신 YOLOv4-tiny 모델에도 널리 활용될 수 있을 것으로 기대된다.

Abstract

In recent real-time embedded platforms, YOLOv4-tiny has gained attention for its fast processing speed and high accuracy, however, its application in real-time environments remains limited due to the constraints of the Systolic Array architecture. To address this, a Fused Layer architecture was proposed, but it is not optimized for the Convolution Operation, Batch Normalization, Leaky ReLU(CBL) block of YOLOv4-tiny, making it difficult to guarantee real-time processing performance. In this paper, we propose a Fused Layer architecture that is suitable for the CBL block. The proposed architecture was synthesized using Synopsys' 14nm open library, resulting in a 296.74% improvement in throughput and a 74.76% reduction in latency compared to the Systolic Array architecture. Therefore, the proposed architecture can contribute to ensuring real-time processing speed for YOLOv4-tiny and is expected to be widely applicable to the latest YOLOv4-tiny models.

Keywords
deep learning, YOLOv4-tiny, fused layer, systolic array, hardware architecture

* 경북대학교 대학원 전자전기공학부 석사과정
- ORCID: https://orcid.org/0009-0002-2738-7463
** 경북대학교 전자공학부/대학원 전자전기공학부 교수 (교신저자)
- ORCID: https://orcid.org/0000-0002-8102-4818

· Received: Sep. 05, 2024, Revised: Sep. 25, 2024, Accepted: Sep. 28, 2024
· Corresponding Author: Byungin Moon
School of Electronics Engineering, Graduate School of Electronic and Electrical Engineering, Kyungpook National University, 80 Daehakro, Bukgu, Daegu 41566, Korea
Tel.: +82-53-950-7580, Email: bihmoon@knu.ac.kr

Ⅰ. 서 론

최근 딥러닝 신경망의 발전으로 객체 검출, 인식 및 세그멘테이션 기술이 자율주행과 같은 실시간 플랫폼에 널리 활용되고 있다[1]-[3]. 이에 따라 실시간 임베디드 플랫폼에서 빠른 처리속도와 높은 정확도가 필수적이며, 이 두 성능 간 최적화를 수행하는 것은 성능 개선의 중요한 요소 중 하나로 고려될 수 있다. 초기에 제안된 신경망 중 Two-Stage 기반의 CNN(Convolution Neural Network) 모델은 후보 영역 제안 단계와 분류 및 식별 단계를 거침으로써 높은 정확도 성능을 달성했지만, 처리량이 증가하여 실시간 처리속도 달성에 한계가 있었다. 이러한 문제를 해결하기 위해 One-Stage CNN 모델이 제안되었으며, 이를 실시간 임베디드 플랫폼에 적용하기 위해 모델의 크기를 줄이면서도 정확도와 처리속도를 최적화하려는 요구가 증가하였다[4][5].

이를 위해 제안된 One-Stage CNN 모델 중 YOLOv4-tiny는 신경망 경량화를 통해 기존 모델 대비 적은 파라미터 수로 매우 빠른 처리속도를 달성하였고, 앵커 박스를 적용하여 다양한 형태 및 크기의 객체에 대한 정확도 성능을 개선하였다[6]. 최근에는 이러한 YOLOv4-tiny를 FPGA와 같은 실시간 임베디드 플랫폼에 적용하기 위해 다양한 하드웨어 구조가 제안되고 있다. 이러한 구조 대부분은 layer-by-layer로 연산을 수행하는 시스톨릭 어레이 구조를 기반으로 구현되는 경향이 있다[7]-[9]. 해당 구조는 하드웨어 자원 재사용을 통해 자원 사용량을 줄임으로써 저면적 및 저비용 구현이 가능하다는 장점이 있다. 그러나 시스톨릭 어레이 구조는 각 layer에서 출력되는 특징맵을 메모리에 접근 및 저장하여 연산을 수행하므로, 파라미터 수, 특징맵의 크기, layer 수 등에 따라 메모리 대역폭 요구치 증가, 처리속도 저하 등 다양한 문제를 초래할 수 있다. 이러한 문제를 해결하기 위한 방법 중 하나로 인접한 layer를 융합하여 연산하는 Fused Layer 구조가 제안되었다[10]. 이 구조는 정확도 성능 저하를 유발하지 않고 인접한 계층을 일부 융합하여 최소한의 특징맵으로 연산함으로써 처리속도를 개선할 수 있다[11][12]. 특히, YOLOv4-tiny는 해당 신경망을 구성하는 모든 layer에서 생성된 특징맵 중 입력 인근 layers에서의 특징맵 크기가 약 50% 이상을 차지하고 있으므로 시스톨릭 어레이 구조를 기반으로 구현하는 것은 처리속도 성능 저하 요인 중 하나로 고려될 수 있다. 그럼에도 불구하고 YOLOv4-tiny에 Fused Layer 방식을 적용하기 위한 연구는 여전히 부족한 실정이며, 다른 CNN과 달리 YOLOv4-tiny의 Convolution Layer는 CBL(Convolution Operation, Batch Normalization, Leaky ReLU) 블록으로 구성되어 있으므로, 이를 고려한 Fused Layer 구조가 요구된다. 이에 본 논문에서는 처리량과 latency 개선을 위해 YOLOv4-tiny의 CBL 블록에 적합한 피라미드 형태의 다중 계층 슬라이딩 윈도우를 위한 일반화된 Fused Layer의 하드웨어 구조를 제안한다.

Ⅱ. 관련 연구

2.1 YOLOv4-tiny

YOLOv4-tiny는 파라미터 수가 약 6M 개로 경량화된 모델이며, 객체 검출에 대한 정확도 및 처리속도를 최적화한 신경망 중 하나이다. 해당 신경망은 백본, 중간부, 헤드로 구성되어 있으며, 신경망의 파라미터 수와 연산량을 줄이기 위해 CSPNet과 OSANet을 결합한 CSPOSANet을 사용하였다[13]. 해당 백본은 파라미터 수와 연산량을 줄이기 위해 설계된 신경망으로, 입력을 두 경로로 나누어 하나는 합성곱 연산을 수행하고 다른 하나는 수행하지 않으며, 이후 두 경로를 다시 합치는 방식으로 동작한다. 이 방식은 자원 사용량을 최적화하면서도 두 경로로 나누어 연산함으로써 효율적인 특징맵 생성이 가능하다. 특히, OSANet은 객체의 위치 정보를 효율적으로 학습하고 객체의 공간적 관계를 강조함으로써 정확한 위치 예측을 가능하게 한다[14]. 이러한 CSPOSANet의 주요 구성 요소는 CBL 블록과 CSP 블록으로 나눌 수 있다. 먼저, 기본적인 CBL 블록은 Convolution, Batch Normalization, Leaky ReLU로 구성되어 있으며, 입력 특징맵에 대해 합성곱 연산 후 배치 정규화와 활성화 함수를 적용하여 최종 특징맵을 출력하는 가장 기본 블록이다.

CSP 블록은 이러한 CBL 블록을 기반으로 Slice, Concatenate, Max Pooling으로 구성되며, 경로를 나누어 연산하고 다시 결합하는 구조를 통해 특징맵의 수용 영역을 개선함으로써 다양한 크기의 객체에 대한 검출 정확도를 높일 수 있다. 중간부는 백본과 헤드 사이에 위치하여 다양한 크기의 특징맵을 통합하고 강조하는 역할을 수행한다. 이를 통해 입력 영상에 존재하는 다양한 크기와 위치를 가지는 객체들이 효율적으로 검출될 수 있다. 특히, YOLOv4-tiny는 경량화된 모델 특성상 백본이 충분히 깊지 않으므로, 중간부에서의 특징맵 통합은 중요한 과정 중 하나이다. 헤드는 Neck에서 전달된 특징맵을 기반으로 객체의 경계 상자와 클래스 정보를 예측한다. 이러한 헤드는 별도의 파라미터(가중치, 편향)를 요구하지 않으므로 연산이 빠르고 효율적이다. 이와 같이 구성함으로써 YOLOv4-tiny는 경량화된 신경망임에도 불구하고 빠른 처리속도와 높은 정확도 성능을 달성함으로써 실시간 객체 검출에 널리 활용되고 있다.

2.2 시스톨릭 어레이 구조

YOLOv4-tiny는 주로 합성곱 연산을 통해 입력 영상으로부터 특징을 추출하며, 이 과정에서 많은 곱셈과 덧셈 연산이 요구된다. 이러한 연산을 자원 효율적으로 수행하기 위한 하드웨어 구조로 시스톨릭 어레이가 널리 활용되고 있다[15]-[18]. 시스톨릭 어레이 구조는 다수의 처리 요소(PE, Processing Element)를 행렬 형태와 유사하게 배치하며, 각 PE는 곱셈기, 가산기, 레지스터로 이루어져 있다[19][20]. 이러한 PE는 가중치의 최대 차원 크기만큼 요구되며, 단순히 PE를 필터 크기에 맞춰 배열하는 것만으로 합성곱 연산을 구현할 수 있으므로 저면적 및 저비용 구현이 가능하다. 이러한 시스톨릭 어레이 구조는 파라미터뿐만 아니라 매 계층(Layer)에서 출력되는 특징맵을 메모리에 접근 및 저장하여 연산을 수행한다. YOLOv4-tiny가 경량화된 모델임에도 불구하고, 매 계층 연산에 요구되는 특징맵 데이터를 메모리에 접근하여 저장하는 과정은 메모리 대역폭 요구치와 처리시간 증가시키는 요인으로 고려할 수 있다. 이에 따라 출력된 특징맵의 크기가 큰 계층에서의 연산량이 크게 증가하여 모델의 추론 연산 시 소요되는 시간에 영향을 줄 수 있다. 특히, YOLOv4-tiny의 경우 표 1과 같이, 특징맵 크기 비중이 입력에 가까운 5개의 계층에 집중되어 있으므로 이를 고려한 최적화가 필요하다. 기존에 제안된 연구에서는 YOLOv4-tiny에 시스톨릭 어레이 구조를 활용함으로써 저면적 및 저비용으로 합성곱 연산의 효율적으로 구현 할 수 있었지만, 특징맵의 높은 연산량과 메모리 접근 및 저장 과정에서 발생하는 처리속도 저하 문제를 해결하기 위한 추가적인 연구가 필요하다.

2.3 Fused Layer 구조

Fused Layer 구조는 layer-by-layer 방식으로 연산하는 시스톨릭 어레이 구조와 달리, 인접한 layer를 융합하여 한 싸이클에 연산함으로써 연산에 소요되는 특징맵의 크기를 최소화하여 메모리 접근 및 사용량과 처리량을 크게 개선할 수 있다[10]. 특히, 신경망에서 특징맵 데이터는 합성곱 계층 전체 데이터(가중치, 편향 등) 중 약 25~50% 이상을 차지할 정도로 그 비중이 크다. 이에 따라 해당 구조에서는 계층을 융합하여 특징맵의 연산 속도를 개선하기 위해 피라미드 형태의 슬라이딩 윈도우를 적용하여 특징맵 데이터를 외부 메모리를 거치지 않고 즉시 처리하여 융합된 마지막 계층의 특징맵을 출력한다. 융합된 계층의 수에 비례하여 계층에서 요구되는 특징맵의 메모리 접근 및 사용량을 최소화하여 처리속도를 개선할 수 있지만, 내부 메모리 사용량과 하드웨어 자원 사용량은 증가할 수 있다. 기존 Fused Layer는 일반적인 합성곱 계층의 융합에 활용된 경향이 있다[21][22]. 특히, 표 1와 같이, 융합된 계층의 수가 2개 정도로 많지 않으며, 그 수가 많을수록 처리속도 증가와 함께 메모리 및 하드웨어 자원 사용량이 비례하여 증가하는 경향이 있다. 이에 따라 CBL 블록을 기반으로 계층이 구성된 YOLOv4-tiny에 기존 방식을 적용하는 것은 적합하지 않을 수 있다. 또한, 초기 계층의 특징맵 비중이 높은 YOLOv4-tiny의 특성을 고려하여 Fused Layer를 적용함으로써 신경망의 처리량과 처리속도 성능이 최적화될 수 있다.

특히, 기존 Fused Layer 구조는 융합된 계층을 처리하는 과정에서 기존 시스톨릭 어레이 구조를 활용하여 연산하는 경향이 있다. 이에 따라 여전히 특징맵의 크기가 큰 Layer에서는 병목현상으로 인해 처리속도 성능 저하가 발생할 수 있다. 이에 따라 YOLOv4-tiny에 최적화된 Fused Layer 구조를 위해 CBL 블록 기반 피라미드 형태의 다중 계층 슬라이딩 윈도우에 관한 연구가 필요하다.

표 1. Fused Layer 클록 싸이클과 외부 메모리 접근 개선 정도
Table 1. Fused layer clock cycles and degree of improvement in external memory access

Degree of improvement	Processing time [%]	External memory access [%]
VGG-16 [12]	about 28	about 47
VGG-16 [21]	about 49	about 29
AlexNet [10]	32.05	28.48

III. 제안하는 하드웨어 구조

3.1 제안하는 방법

YOLO4-tiny는 CBL 블록 외에도 풀링 계층, 분리(Slice) 계층, 결합(Concatenate) 계층을 포함하고 있다. 이러한 계층들은 표 2와 같은 순서와 차원으로 구성되어 있으며, 총 39개의 계층이 존재한다. 표 2는 연산 정밀도가 16비트인 Half Precision일 때를 기준으로 하며, 각 계층에서 출력되는 특징맵의 3차원 크기를 나타낸 것이다. 또한, YOLOv4-tiny에서 CBL은 합성곱 연산, 배치 정규화(BN), Leaky ReLU를, MP는 Max Pooling, Concat은 결합 계층을 의미한다. YOLOv4-tiny의 계층 구성은 입력단의 특징맵 크기가 매우 크기 때문에 MP를 적용하여 특징맵의 2차원 크기를 절반으로 줄인다. 그럼에도 불구하고 CBL 1~3 블록의 출력 특징맵이 차지하는 비중은 전체 특징맵 중 44.03%이고, MP 1~2 계층을 포함할 경우 54.19%로 5개의 계층의 특징맵이 절반 이상을 차지한다.

표 2. YOLOv4-tiny의 계층별 특징맵 크기와 비중
Table 2. Feature map sizes and proportions per layer of YOLOv4-tiny

Layer name	Feature map channel	Feature map size [bit]	Portion [%]
Input	416×416×3	8,306,688	0.03
CBL 1	416×416×32	88,604,672	27.09
MP 1	208×208×32	22,151,168	6.77
CBL 2	208×208×64	44,302,336	13.55
MP 2	104×104×64	11,075,584	3.39
CBL 3	104×104×64	11,075,584	3.39
Slice 1	104×104×32	5,537,792	1.69
CBL 4	104×104×32	5,537,792	1.69
CBL 5	104×104×32	5,537,792	1.69
Concat 1	104×104×64	11,075,584	3.39
CBL 6	104×104×64	11,075,584	3.39
Concat 2	104×104×128	11,075,584	6.77
MP 3	52×52×128	2,768,896	0.85
CBL 7	52×52×128	5,537,792	1.69
Slice 2	52×52×64	2,768,896	0.85
CBL 8	52×52×64	2,768,896	0.85
CBL 9	52×52×64	2,768,896	0.85
Concat 3	52×52×128	5,537,792	1.69
CBL 10	52×52×128	5,537,792	1.69
Concat 4	52×52×256	11,075,584	3.39
MP 4	26×26×256	2,768,896	0.85
CBL 11	26×26×256	2,768,896	0.85
Slice 3	26×26×128	1,384,448	0.42
CBL 12	26×26×128	1,384,448	0.42
CBL 13	26×26×128	1,384,448	0.42
Concat 5	26×26×256	2,768,896	0.85
CBL 14	26×26×128	1,384,448	0.42
Concat 6	26×26×256	2,768,896	0.85
CBL 15	26×26×256	2,768,896	0.85
Concat 7	26×26×512	5,537,792	1.69
MP 5	13×13×512	1,384,448	0.42
CBL 16	13×13×512	1,384,448	0.42
CBL 17	13×13×256	692,224	0.21
CBL 18	13×13×512	1,384,448	0.42
CBL 19	13×13×255	689,520	0.21
CBL 20	13×13×128	346,112	0.11
Scaling	26×26×128	1,384,448	0.42
Concat 8	26×26×384	4,153,344	1.27
CBL 21	26×26×256	2,768,896	0.85
CBL 22	26×26×255	2,758,080	0.84

이에 따라 이 5개의 계층에서 연산에 요구되는 특징맵의 크기를 최소화하기 위한 Fused Layer 구조를 제안한다. 해당 신경망에서는 모든 가중치는 3×3 크기의 행렬이며, 3, 4차원의 크기는 계층마다 일부 상이하다.

먼저, 416×416×3 크기의 입력 특징맵에서 18×18×3 크기만큼만 추출하여 CBL 3의 출력 특징맵에서 1×1×64 크기의 행렬을 최종 출력한다. 입력 특징맵으로부터 18×18×3 크기를 잘라내어 CBL 1 블록을 거치는 경우 그 특징맵의 크기는 16×16×32이며, MP 1을 통해 8×8×32 크기로 축소한다. 그 다음 CBL 2 블록을 거쳐 6×6×64 크기의 특징맵을 출력하고 이에 대해 MP 2를 거쳐 3×3×64 크기로 축소한다. CBL 3 블록을 거치면 1×1×64 크기의 특징맵이 출력되며, 이 과정을 104×104 만큼 반복함으로써 CBL 3의 출력 특징맵을 얻을 수 있다. 이러한 연산을 일반화하기 위해 다음과 같이 행렬을 고려한다.

$$W^{Ch}=\begin{pmatrix} w_{11}^{ch} & w_{12}^{ch} & w_{13}^{ch} \\ w_{21}^{ch} & w_{22}^{ch} & w_{23}^{ch} \\ w_{31}^{ch} & w_{32}^{ch} & w_{33}^{ch} \end{pmatrix} \quad (1)$$

CBL 블록을 위한 피라미드 형태의 다중 계층 슬라이딩 윈도우의 하드웨어 구조를 일반화하기 위해 가중치를 식 (1)과 같이 행렬 W를 정의한다. 행렬 W에서 ch는 채널을 의미하고, 이전 계층의 출력 특징맵의 크기가 k×k×ch라고 가정할 때 이 특징맵으로부터 추출한 행렬 N을 식 (2)와 같이 가정한다. Ch는 k×k×ch에서 채널인 ch를 의미하며, 식 (2)에서 (n,m)은 k×k×ch 특징맵에서 임의의 위치에 대한 3×3 크기의 행렬을 의미한다.

$$N_{(n,m)}^{Ch}=\begin{pmatrix} a_{(n,m)}^{ch} & a_{(n,m+1)}^{ch} & a_{(n,m+2)}^{ch} \\ a_{(n+1,m)}^{ch} & a_{(n+1,m+1)}^{ch} & a_{(n+1,m+2)}^{ch} \\ a_{(n+2,m)}^{ch} & a_{(n+2,m+1)}^{ch} & a_{(n+2,m+2)}^{ch} \end{pmatrix} \quad (2)$$

이때 n과 m의 최대값은 (k-2)이며, 식 (2)는 k×k 행렬에 대한 CBL 블록의 합성곱 연산을 한 번에 처리하기 위해 일반화한 것이다. 예를 들어, 5×5 행렬이 제시되었을 때, $N_{(1,1)}^{1}$의 5×5 행렬의 (1,1) 위치를 기준으로 3×3 크기에 해당하는 커널을 의미한다. 이와 동일하게 $N_{(3,3)}^{1}$는 5×5 행렬의 (5,5) 위치를 기준으로 하는 커널이다. 이와 같이 k×k×ch 행렬에 대해 각 채널에 동일 위치에 존재하는 $N_{(n,m)}^{Ch}$ 커널을 채널 Ch 만큼 생성하고, 이 모두를 가중치 W^{Ch} 와 대응하는 원소 간 곱셈만 수행하며 이 과정에서 1 clk(1 clock cycle)이 소요된다. 이후, 이렇게 생성된 원소들 간 덧셈이 2 clk에서 연산되고, 채널 Ch 만큼 생성된 모든 값과 bias(편향)을 모두 더하고 3 clk에서 진행된다. 4 clk에서 모든 덧셈 연산이 완료된 값은 CBL의 활성함수인 Leaky ReLU를 거쳐 최종적으로 출력된다. 이와 같은 수식을 기반으로 하여, 18×18×3 행렬에 대해 CBL 블록의 연산을 수행하는 하드웨어 구조는 그림 1과 같이 나타낼 수 있다.

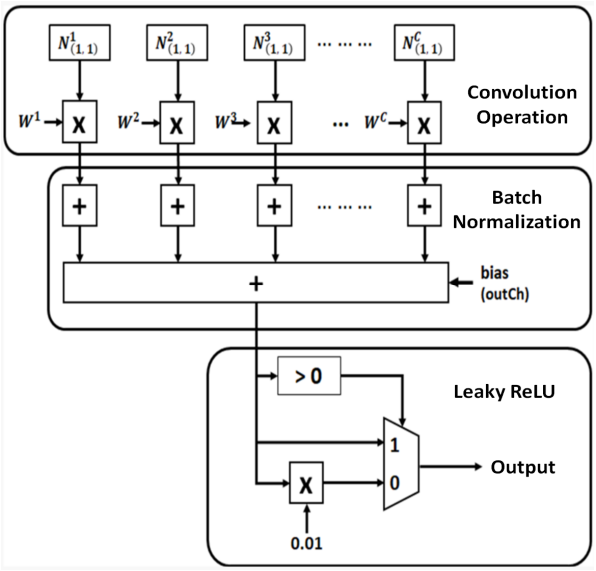

그림 1. $N_{(n,m)}^{Ch}$ 커널 연산을 위한 CBL 블록의 제안하는 하드웨어 구조

Fig. 1. Proposed hardware architecture of CBL block for $N_{(n,m)}^{Ch}$ Kernel operations

CBL 블록의 연산을 수행하는 이 구조는 3개의 채널에 대해 각각 3×3 커널을 생성하며, 그 개수(length)는 각 채널당 총 256개다. 동일한 위치에 있는 커널 3쌍이 합성곱 모듈로 입력되고 각 채널에 해당하는 가중치와 원소 간 곱셈을 수행한다. 이후 이들 간 덧셈을 수행하여 3개의 합성곱 결과를 생성하고, 이들 값은 편향 값과 함께 더하여 활성 함수를 통과함으로써 출력한다. 이들 과정은 모두 병렬화하여 한 번에 출력되고, 16×16×1 크기의 특징맵을 생성한다. 이러한 과정은 CBL 1의 outCh인 32만큼 수행해야 하며, 본 구조에서는 파이프라이닝을 통해 첫 16×16×1이 출력되는 시점부터 outCh에 대한 연산이 진행되고, 16채널씩 병렬화하여 하드웨어 구조를 확립할 수 있다.

이와 달리, 기존에 제안된 Fused Layer 구조에서는 계층을 융합하는 과정에서 추출된 특징맵을 처리하기 위해 시스톨릭 어레이 구조로 연산하게 된다. 이에 따라 융합된 계층들의 연산이 완료되면, 다음 출력을 위해 피라미드 형태의 슬라이딩 윈도우를 통해 이중으로 연산을 수행하므로, 입력 특징맵의 크기가 클수록 처리속도가 크게 느려질 수 있다. 이를 해결하기 위해 제안하는 방법에서는 시스톨릭 어레이 구조를 사용하지 않고, 융합된 각 계층에서의 연산을 한 싸이클에 수행함으로써 단위시간당 출력되는 비트 양을 대폭 개선할 수 있다.

3.2 제안하는 하드웨어 구조

YOLOv4-tiny에 Fused Layer를 적용하기 위한 CBL 블록 기반 피라미드 형태의 다중 계층 슬라이딩 윈도우는 식 (1)과 식 (2)를 통해 일반화할 수 있다. 먼저, MP의 하드웨어 구조는 그림 2와 같이 MUX(Multiplexer)를 활용하여 단순하게 나타낼 수 있다. 해당 신경망에서 MP는 2×2 행렬을 기본으로 출력할 전체 채널 수만큼 사용된다. 2×2 행렬 연산을 기준으로 설명하면, 4개의 원소 값들 중 2개를 하나의 그룹으로 묶어 각 그룹(a와 c, b와 d)에 대해 크기 비교를 수행한다. 그 중 큰 값(a, b)을 통과시키며, 통과된 값들(a는 x, b는 y) 간 재비교를 통해 결과적으로 4개의 값 중 가장 큰 값(x)을 출력하는 형태이다. CBL 블록에 포함된 합성곱 연산은 식 (2)와 같은 행렬을 생성하여 수행한다.

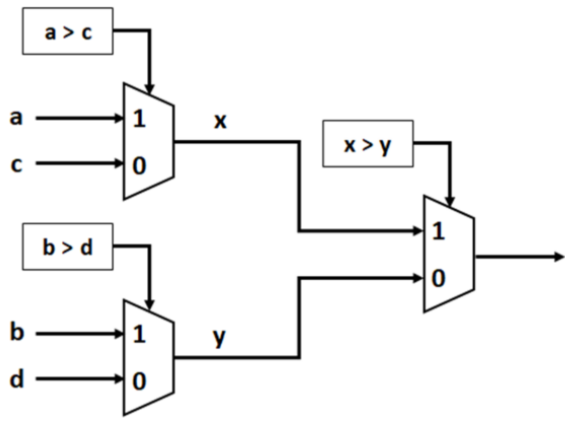

그림 2. MP 연산을 위한 제안하는 하드웨어 구조
Fig. 2. Proposed hardware architecture for MP operations

시스톨릭 어레이 구조에서 특징맵을 가중치 필터 크기만큼 입력받아 매 클록 싸이클 마다 처리하지만, 제안하는 구조에서는 처리해야 할 행렬을 한 싸이클에 생성한 뒤에 처리함으로써 처리량을 획기적으로 증가시키고 이를 통해 latency를 크게 개선할 수 있다.

N×N 크기의 2차원 입력 특징맵은 채널 Ch 만큼 존재하며, 이를 N×N×Ch와 같이 표현한다. CBL 블록에서는 가중치가 3×3×Ch의 차원을 가지며, Ch 값이 동일한 특징맵과 합성곱 연산을 수행한다. 이후 합성곱 연산이 종료되어 Ch 크기만큼의 스칼라값이 생성되며, 이들 모두 더함으로써 배치 정규화(BN)가 진행된다. 이렇게 생성된 값은 CBL 블록의 활성화함수인 Leaky ReLU를 통과하여 최종값을 출력한다. 이러한 과정을 시스톨릭 어레이 구조는 특징맵 전체를 가중치가 래스터 스캐닝(Raster scanning)하여 출력하지만, 제안하는 구조에서는 래스터 스캐닝 시 요구되는 합성곱 횟수 만큼의 행렬을 미리 생성하여 합성곱을 수행하고, 이렇게 생성된 값들에 BN을 수행하고자 다음 클록에서 더한다. 그리고 BN이 완료된 값에 편향(bias)을 더하여 Leaky ReLU를 통과함으로써 최종값을 출력하게 된다. 이를 일반화한 구조는 그림 3과 같다. 식 (1)에서 정의한 가중치 행렬과 입력 특징맵으로부터 추출한 식 (2)의 행렬은 합성곱 연산을 위해 대응되는 원소 간 곱셈을 수행하고, 이들 값을 모두 더하여 다음 가산기로 입력된다. 이후 Ch 만큼 생성된 합성곱 결과들을 모두 더하며, 이때 편향 값도 함께 더함으로써 BN이 완료된다. 이 값이 Leaky ReLU를 통과할 때 0 이상은 통과하고 그 외 값은 0.01 값을 곱하여 출력한다. 이러한 일반화된 CBL 블록 구조는 CBL 1 ~ 3에 모두 적용 가능하며, YOLOv4-tiny를 구성하는 모든 CBL 블록에 활용될 수 있다.

이는 표 3과 같이 해당 구조의 크기를 결정짓는 파라미터를 통해 다른 CBL 블록에도 적용이 가능하다. N×N×Ch 크기의 특징맵의 경우, 단 하나의 채널에 Fused Layer를 적용하기 위해 필요한 행렬의 길이 L은 2차원 길이에 2를 뺀, (N-2)×(N-2)이며, 이 값이 전체 채널 수 Ch 만큼 필요하다. 제안하는 구조는 표 3의 계층명에 기입된 5개의 계층에 모두 적용한다.

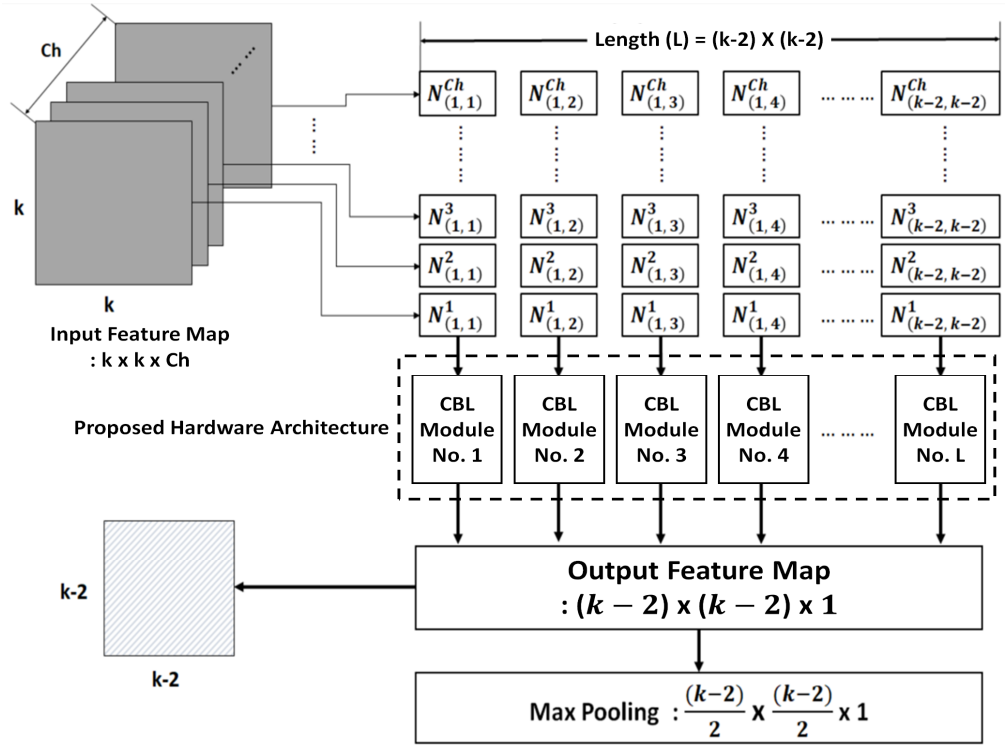

그림 3. 피라미드 형태의 다중계층 슬라이딩 윈도우를 위한 일반화된 하드웨어 구조
Fig. 3. Generalized hardware architecture of CBL block for pyramid shaped multi-layer sliding window

표 3. 일반화된 제안하는 하드웨어 구조의 파라미터들
Table 3. Parameters of the generalized proposed hardware architecture

Layer name	Feature map channel (N x N x Ch)	Length (L)	outCh	Ch
CBL 1	18 × 18 × 3	256	1 ~ 32	3
MP 1	16 × 16 × 32	-	-	-
CBL 2	8 × 8 × 32	36	1 ~ 64	32
MP 2	6 × 6 × 64	-	-	-
CBL 3	3 × 3 × 64	1	1 ~ 64	64
Output	1 × 1 × 64	-	-	-

MP의 경우, 그림 2와 같은 널리 활용되는 구조를 활용하며, 나머지 3개의 CBL 블록에는 제안하는 구조를 적용한다.

CBL 3의 출력 특징맵은 104×104×64 크기이며, 이 특징맵에서 1×1×64 크기의 특징맵을 한 싸이클에 출력하기 위해 일반화된 구조를 활용하여 피라미드 형태의 다중계층 슬라이딩 윈도우를 설계하였다. 해당 윈도우는 416×416×3 크기의 특징맵에 래스터 스캐닝을 수행함으로써 CBL 3의 출력 특징맵을 얻을 수 있다. 이때 래스터 스캐닝은 최종 출력 특징맵을 기준으로 산출해야 한다. CBL 3에서 2차원 크기의 특징맵을 기준으로 (1,1) 위치의 값을 출력한 뒤에 다음 값은 (1,2)를 출력할 수 있는 값을 입력 특징맵으로부터 추출하여 연산을 수행한다. 표 3의 계층명과 같이 CBL 1을 시작으로 MP 1, CBL 2, MP 2, CBL 3를 융합하고, 특징맵 채널에 표기된 크기의 특징맵에 대해 식 (2)를 적용한 행렬을 생성하여 연산을 수행한다. CBL 블록의 연산은 그림 1과 같고 모든 연산에 4 clk cycles이 소요되며 MP 연산의 경우 한 채널당 1 clk cycle이 소요되도록 설계한다. 또한, 연산에 소요되는 clk cycle을 최소화하기 위해 채널 수를 절반으로 나누어 병렬화하여 연산한다. 이에 따라 각 계층이 연산 시작은 1, 5, 21, 25, 57 clk cycle이며, 가장 마지막에 출력되는 CBL 3의 (104, 104) 위치의 값은 91 clk cycles이 지난 후에 얻을 수 있다. 결과적으로 CBL 블록을 기반으로 하는 피라미드 형태의 다중계층 슬라이딩 윈도우를 사용하여 Fused Layer를 적용하는 경우, 5개의 융합 계층을 연산하는데 총 205,577 clk cycles이 소요된다.

Ⅳ. 실험 결과

YOLOv4-tiny에 Fused Layer를 적용하고자 CBL 블록을 기반으로 하는 피라미드 형태의 다중계층 슬라이딩 윈도우를 제안했다. 해당 윈도우는 특징맵의 비중이 가장 큰 계층 5개에 대해 적용하였으며, 이를 시스톨릭 어레이 구조와 비교한다. 먼저, 시스톨릭 어레이 구조는 가중치를 고정하는 Weight Stationary 방식으로 설계하였으며, 이 과정에서 처리속도 최소화를 위해 MP 연산은 합성곱 연산 중 종료되도록 고려하였다. 시스톨릭 어레이 구조와 제안하는 구조는 모두 Xilinx Vivado 2023.2에서 HDL을 기반으로 설계되었으며, 해당 구조의 타겟 칩셋으로 Virtex UltraScale+ XCVU19P-FSVA3824-1-E를 선정하여 시뮬레이션을 통한 동작 검증을 수행하였다. 동작 검증의 경우, 입력 영상에 대한 처리 결과를 테스트 벤치를 통해 특징맵을 출력하였다. 이러한 특징맵은 단순히 실수의 배열이므로, MATLAB을 통해 출력한 특징맵과 대응되는 위치의 실수 값들을 일치 정도를 파악함으로써 동작을 검증하였다. 해당 구조는 저전력, 고성능 설계를 위해 Synopsis사의 14nm 오픈 라이브러리(saed14rvt_ss0p72v125c)와 Design Compiler를 사용하여 합성하였다. 또한, SS(Slow-Slow)와 RVT(Regular Voltage Threshold) 옵션을 적용하여 성능과 전력 소비의 균형을 유지하였으며, 최악의 성능 조건에서도 회로가 안정적으로 동작하도록 고려하였다.

표 4. 시스톨릭 어레이와 제안하는 구조 간 면적과 전력 비교
Table 4. Area and power comparison between systolic array and proposed architecture

Cell area	Systolic array	Fused layer	Comparison
Convolution module	187,981.33	1,567,819.40	+734.03%
Max pooling module	3,868.83	37,550.50	+870.58%
Total area	191,850.17	1,605,369.46	+736.78%
Power [mW]	4.21	44.27	+950.55%

또한, 0.72V 전압 조건과 125°C의 고온 환경을 설정하여 저전력 고온 환경에서의 동작을 최적화하였다. 이러한 설정을 통해 다양한 환경에서의 동작 안정성을 확보하고자 하였으며, 이를 통해 설계의 성능, 전력 소비량, 신뢰성을 극대화하기 위해 오픈 라이브러리를 선택하여 합성하였다. 다음의 표 4와 표 5는 면적 및 전력과 타이밍 분석 등을 비교하여 정리한 것이다.

이 표에서의 비교값은 시스톨릭 어레이 대비 제안하는 구조의 증감율을 표시한 것이다. 예를 들어 합성곱 모듈에서 제안하는 구조는 시스톨릭 어레이 구조의 734.03% 수준으로 증가하였음을 의미한다. 시스톨릭 어레이는 합성곱 모듈과 MP 모듈로 이루어져 있으며, 해당 구조는 하드웨어 자원 재사용을 통해 저면적 구현이 가능하다. 이에 따라 표 4과 같이 제안하는 구조 대비 매우 적은 수준의 셀 면적을 차지한다. 그에 비해 제안하는 구조는 모든 계층에 대한 모듈을 요구하므로 연결된 계층이 많을수록 첫 계층 연산에 요구되는 하드웨어 자원이 매우 높은 것을 알 수 있다. 또한, 이에 따라 전력 소비량이 시스톨릭 어레이 대비 Fused Layer 구조에서 큰 폭으로 증가할 수있다. 그러나 제안하는 구조는 표 5와 같이 Critical Path Time과 최대 동작 가능 주파수가 낮음에도 불구하고 연산에 소요되는 clk cycles 수와 latency를 매우 큰 폭으로 개선할 수 있다.

표 5. 시스톨릭 어레이와 제안하는 구조 간 타이밍 분석, 레이턴시, 그리고 처리량 비교
Table 5. Timing analysis, latency, and throughput comparison between systolic array and proposed architecture

Timing analysis	Systolic array	Fused layer	Comparison
Critical path time [ns]	4.977	6.644	+33.50%
Max. clock frequency [MHz]	200.9	150.5	-25.09%
Clock cycles	229,320	43,355	+81.09%
Latency [ms]	1.141	0.288	+74.76%
Throughput [bps]	9.660 G	38.323 G	-296.74%

특히, 합성곱 연산을 위한 행렬을 모두 생성하여 연산함으로써 초당 처리할 수 있는 비트 크기가 매우 큰 폭으로 개선된다는 것을 확인할 수 있다. 이와 같은 합성 결과를 통해, CBL 블록에 적합한 제안하는 구조는 셀 면적을 736.78% 더 많이 사용하지만, 처리량이 296.74% 만큼 개선되므로 Latency가 74.76% 개선됨을 알 수 있다. 또한, 최대 동작 가능 주파수는 비교적 낮음에도 불구하고 클록 싸이클의 경우 기존 대비 매우 낮은 수준으로도 구현할 수 있다.

표 6. 시스톨릭 어레이와 제안하는 구조 간 면적과 전력 비교
Table 6. Latency and throughput comparison of the entire YOLOv4-tiny model on the systolic array architecture

	[22]	[23]	[24]
Max. clock frequency [MHz]	-	100	200
Clock cycles	-	-	247,368
Latency [ms]	376	9.08	-
Throughput [bps]	9.240 G	74.45 G	24.252 G

또한, 시스톨릭 어레이 구조를 기반으로 하는 신경망인 YOLOv4-tiny는 표 6과 같은 경향성을 가진다. 이를 통해 알 수 있는 것은 YOLOv4-tiny에서 약 50% 이상을 차지하는 초기 계층의 연산에 대해 제안하는 방법은 매우 빠른 처리속도로 연산이 가능하다는 점이다. 특히, 연산에 소요되는 클록 싸이클은 특징맵이 차지하는 비율이 높지만 이러한 수치에 비해 요구되는 수가 매우 낮다는 것을 확인할 수 있다. 그러므로 제안하는 구조를 YOLOv4-tiny를 실시간 임베디드 플랫폼에 구현하는 경우, 특징맵의 비중이 높은 입력 계층에 적용함으로써 해당 계층들에서 요구되는 처리량과 Latency가 크게 개선되어 실시간 처리속도 보장에 기여할 수 있다.

Ⅴ. 결론 및 향후 과제

본 논문에서는 YOLOv4-tiny의 처리량과 latency를 개선하기 위해 CBL 블록을 고려한 일반화된 Fused Layer 구조를 제안하였다. YOLOv4-tiny는 경량화된 구조로 객체 검출을 위한 실시간 임베디드 플랫폼에 널리 활용되고 있으나, 기존 시스톨릭 어레이 구조는 저면적 및 저비용 구현에 초점을 맞추어 처리속도가 낮은 경향이 있었다. 이를 극복하기 위해 Fused Layer 구조가 제안되었으나 YOLOv4-tiny를 구성하는 CBL 블록의 특성에 최적화되지 않아 처리속도 개선에 한계가 있었다.

이에 따라 본 논문에서 제안하는 구조는 CBL 블록 기반 피라미드 형태의 다중계층 슬라이딩 윈도우 방법을 도입하여 YOLOv4-tiny의 모든 계층에 Fused Layer 구조를 적용할 수 있는 일반화된 구조를 제안하였다. 특히, YOLOv4-tiny는 특징맵의 크기가 입력단에 집중되어 있어 초기 5개 계층에 대해 제안하는 구조를 적용함으로써 처리량과 latency 성능 개선 정도를 측정하였다. 제안하는 구조는 Synopsys의 14nm 오픈 라이브러리와 Design Compiler를 사용하여 합성되었으며, RVT(Regular Voltage Threshold)와 SS(Slow-Slow) 옵션을 적용하여 성능과 전력 소비의 균형을 유지하였다. 또한, 0.72V 전압 조건과 125°C의 고온 환경에서도 안정적인 동작을 보장하도록 설계되었다. 이러한 환경에서 실험한 결과, 제안하는 구조는 기존 시스톨릭 어레이 구조 대비 처리량이 296.74% 향상되고, latency가 74.76% 감소하는 것을 확인하였다. 기존 Fused Layer 관한 연구는 CBL 블록에 최적화되어 있지 않아 YOLOv4-tiny를 구성하는 임의의 계층에 적용하는 것이 도전적이었다.

본 논문에서 제안하는 구조는 이러한 한계를 극복하여 YOLOv4-tiny의 모든 계층에 적용 가능한 일반화된 구조로 설계되었다. 이러한 구조는 YOLOv4-tiny의 초기 계층 연산에 소요되는 시간을 개선하여 신경망 모델의 추론 시간을 최적화하였다. 최근에 제안되고 있는 YOLOv4-tiny는 지속적으로 경량화되고 있으며, 이로 인해 파라미터와 계층 수가 감소하여 특징맵의 크기가 줄어들고 있다. 이에 따라 경량화된 최신 YOLOv4-tiny 모델에 제안하는 구조를 적용할 경우, 더 적은 면적으로 처리속도 개선이 가능할 수 있다. 이에 따라 제안하는 구조는 자율주행, 의료 응급 진단, 군사 및 보안 시스템 등과 같은 빠르고 정확한 의사결정이 필수적인 분야에 활용할 경우, 신뢰성과 안정성을 확보하고 높은 처리속도 보장에 기여할 수 있을 것으로 예상된다.

향후 연구에서는 저면적 및 저비용 환경에서 실시간 성능을 보장할 수 있도록 제안하는 구조의 처리속도는 유지하면서 요구되는 면적을 최적화할 필요가 있다.

References

[1] V. Viswanatha, R. K. Chandana, and A. C. Ramachandra, "Real time object detection system with YOLO and CNN models: A review", arXiv Prepr. arXiv2208.00773, Vol. 14, pp. 144-151, Jul. 2022. https://doi.org/10.48550/arXiv.2208.00773.

[2] Z. Ouyang, J. Niu, Y. Liu, and M. Guizani, "Deep CNN-Based Real-Time Traffic Light Detector for Self-Driving Vehicles", IEEE Transactions on Mobile Computing, Vol. 19, No. 2, pp. 300-313, Feb. 2020. https://doi.org/10.1109/TMC.2019.2892451.

[3] M. Maity, S. Banerjee, and S. S. Chaudhuri, "Faster r-cnn and yolo based vehicle detection: A survey", 2021 5th international conference on computing methodologies and communication (ICCMC), Erode, India, pp. 1442-1447, Apr. 2021. https://doi.org/10.1109/ICCMC51019.2021.9418274.

[4] S. Zhang, et al., "An fpga-based reconfigurable cnn accelerator for yolo", 2020 IEEE 3rd international conference on electronics technology (ICET), Chengdu, China, pp. 74-78, May 2020. https://doi.org/10.1109/ICET49382.2020.9119500.

[5] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection", Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, Seattle, WA, USA, pp. 10781-10790, Jun. 2020. https://doi.org/10.48550/arXiv.1911.09070.

[6] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-yolov4: Scaling cross stage partial network", Proc. of the IEEE/cvf conference on computer vision and pattern recognition, Nashville, TN, USA, pp. 13029-13038, Jun. 2021. https://doi.org/10.1109/cvpr46437.2021.01283.

[7] J. Shen and Y. Zhou, "Accurate and real-time object detection in crowded indoor spaces based on the fusion of DBSCAN algorithm and improved YOLOv4-tiny network", Journal of Intelligent Systems, Vol. 32, No. 1, Jul. 2023. https://doi.org/10.1515/jisys-2022-0268.

[8] S. Li, Y. Luo, K. Sun, N. Yadav, and K. K. Choi, "A Novel FPGA Accelerator Design for Real-Time and Ultra-Low Power Deep Convolutional Neural Networks Compared With Titan X GPU", IEEE Access, Vol. 8, pp. 105455-105471, Jun. 2020. https://doi.org/10.1109/access.2020.3000009.

[9] L. Liu and S. Brown, "Leveraging Fine-grained Structured Sparsity for CNN Inference on Systolic Array Architectures", 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), Dresden, Germany, pp. 301-305, Aug. 2021. https://doi.org/10.1109/fpl53798.2021.00060.

[10] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators", 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, pp. 1-12, Oct. 2016. https://doi.org/10.1109/micro.2016.7783725.

[11] M. Li, N. Wang, H. Zhou, Y. Duan, and J. Wu, "Fused-Layer-based DNN Model Parallelism and Partial Computation Offloading", GLOBECOM 2022-2022 IEEE Global Communications Conference, Rio de Janeiro, Brazil, pp. 5195-5200, Dec. 2022. https://doi.org/10.1109/globecom48099.2022.10000779.

[12] F. Indirli, A. Erdem, and C. Silvano, "A Tile-based Fused-layer CNN Accelerator for FPGAs", 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Glasgow, UK, pp. 1-4. Nov. 2020. https://doi.org/10.1109/icecs49266.2020.9294981.

[13] C.-Y. Wang, H.-Y. Mark Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN", Proc. of the IEEE/CVF conference on computer vision and pattern recognition workshops, Seattle, WA, USA, pp. 390-391, Jun. 2020. https://doi.org/10.1109/cvprw50498.2020.00203.

[14] Y. Lee, J. Hwang, S. Lee, Y. Bae, and J. Park, "An Energy and GPU-Computation Efficient Backbone Network for Real-Time Object Detection", Proc. of the IEEE/CVF conference on computer vision and pattern recognition workshops. Long Beach, CA, USA, pp. 752-760, Jun. 2019. https://doi.org/10.1109/cvprw.2019.00103.

[15] Z. Valadanzoj, H. Daryanavard, and A. Harifi, "High-speed YOLOv4-tiny hardware accelerator for self-driving automotive", The Journal of Supercomputing, Vol. 80, No. 5, pp. 6699-6724, Oct. 2023. https://doi.org/10.1007/s11227-023-05713-2.

[16] V. Jain, N. Jadhav, and M. Verhelst, "Enabling real-time object detection on low cost FPGAs", Journal of Real-Time Image Processing, Vol. 19, No. 1, pp. 217-229, Oct. 2021. https://doi.org/10.1007/s11554-021-01177-w.

[17] D. Pestana, et al., "A Full Featured Configurable Accelerator for Object Detection With YOLO", IEEE Access, Vol. 9, pp. 75864-75877, May 2021. https://doi.org/10.1109/access.2021.3081818.

[18] K. Lim, G. Kim, T. Park, X. T. Nguyen, and H.-J. Lee, "An Energy-Efficient YOLO Accelerator Optimizing Filter Switching Activity", 2022 IEEE International Symposium on Circuits and Systems (ISCAS), Austin, TX, USA, pp. 2472-2476, May 2022. https://doi.org/10.1109/iscas48785.2022.9937904.

[19] G. Zhang, R. Zhang, R. Wang, and S. Zhu, "A Systolic Array-Based Scheduling Strategy for Sparse CNN Accelerators", IEEE Transactions on Circuits and Systems II: Express Briefs, Vol. 71, No. 3, pp. 1436-1440, Mar. 2024. https://doi.org/10.1109/tcsii.2023.3326489.

[20] C. Yang, Y. Yang, W. Yang, L. Huang, and Y. Li, "A General-Purpose CNN Accelerator Based on Improved Systolic Array for FPGAs", 2022 7th International Conference on Integrated Circuits and Microsystems (ICICM), Xi'an, China, pp. 529-533, Oct. 2022. https://doi.org/10.1109/icicm56102.2022.10011386.

[21] A. Erdem, D. Babic, and C. Silvano, "A Tile-based Fused-Layer Approach to Accelerate DCNNs on Low-Density FPGAs", 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Genoa, Italy, pp. 37-40, Nov. 2019. https://doi.org/10.1109/icecs46596.2019.8964870.

[22] S. Xu, Y. Zhou, Y. Huang, and T. Han, "YOLOv4-tiny-based coal gangue image recognition and FPGA implementation", Micromachines, Vol. 13, No. 11, pp. 1-19, Nov. 2022. https://doi.org/10.3390/mi13111983.

[23] M. Kim, et al. "A Low-Latency FPGA Accelerator for YOLOv3-Tiny With Flexible Layerwise Mapping and Dataflow", in IEEE Transactions on Circuits and Systems I: Regular Papers, Vol. 71, No. 3, pp. 1158-1171, Mar. 2024. https://doi.org/10.1109/TCSI.2023.3335949.

[24] Q. Song, J. Zhang, L. Sun, and G. Jin, "Design and Implementation of Convolutional Neural Networks Accelerator Based on Multidie", in IEEE Access, Vol. 10, pp. 91497-91508, Aug. 2022. https://doi.org/10.1109/ACCESS.2022.3199441.

저자소개

김 경 욱 (Kyeonguk Kim)

2021년 8월 : 창원대학교
전기공학과(공학사)
2024년 8월 : 경북대학교
전자전기공학부(공학석사)
관심분야 : SoC, 디지털 VLSI, 컴퓨터 비전

문 병 인 (Byungin Moon)

1995년 2월 : 연세대학교
전자공학과(공학사)
1997년 2월 : 연세대학교
전자공학과(공학석사)
2002년 2월 : 연세대학교
전기전자공학과(공학박사)
2002년 2월 ~ 2004년 3월 :
하이닉스반도체 선임연구원
2004년 4월 ~ 2005년 1월 : 연세대학교 BK 사업단
연구교수
2005년 2월 ~ 현재 : 경북대학교 전자공학부/대학원
전자전기공학부 교수
관심분야 : SoC, 컴퓨터 구조, 비전 프로세서