(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2025-0064654
(43) 공개일자 2025년05월09일

(51) 국제특허분류(Int. Cl.)
G06N 3/065 (2023.01) *G06F 15/78* (2006.01)
G06F 17/15 (2006.01) *G06N 3/0455* (2023.01)
G06N 3/0464 (2023.01)
(52) CPC특허분류
G06N 3/065 (2023.01)
G06F 15/781 (2013.01)
(21) 출원번호 10-2024-0196300(분할)
(22) 출원일자 2024년12월24일
심사청구일자 없음
(62) 원출원 특허 10-2023-0146032
원출원일자 2023년10월27일
심사청구일자 2023년10월27일

(71) 출원인
세종대학교산학협력단
서울특별시 광진구 능동로 209 (군자동, 세종대학교)
(72) 발명자
박우찬
서울특별시 광진구 능동로 209, 세종대학교 대양AI센터 723호 (군자동)
이진영
서울특별시 광진구 능동로 209, 대양AI센터 827호(군자동, 세종대학교)
(뒷면에 계속)
(74) 대리인
정부연

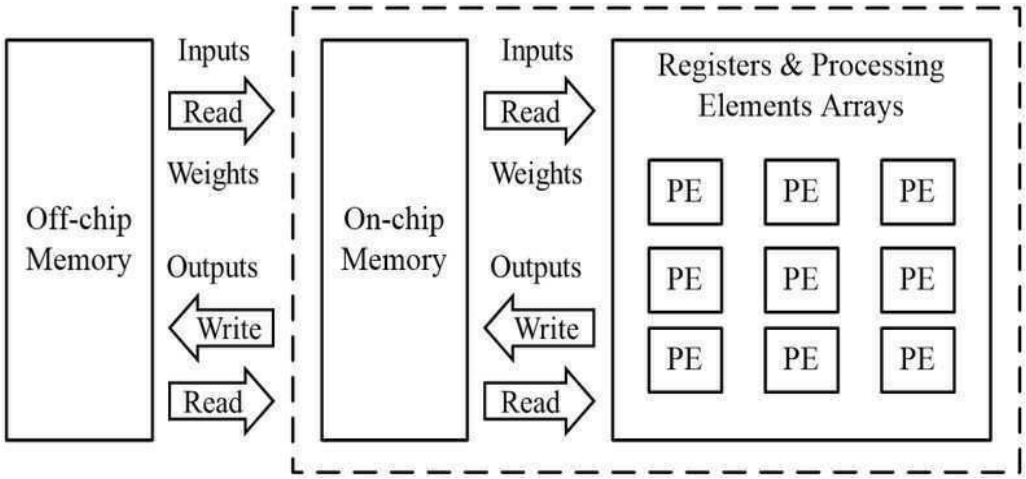
전체 청구항 수 : 총 12 항

(54) 발명의 명칭 **메모리 참조에 효과적인 실시간 인공지능 가속 하드웨어 장치 및 그 동작 방법**

(57) 요 약

본 발명은 메모리 참조에 효과적인 실시간 인공지능 가속 하드웨어 장치 및 그 동작 방법에 관한 것으로, 상기 장치는 신경망의 동작을 구현하기 위하여 신경망 구조에 따라 연결된 신경망 유닛(Unit)들; 오프칩 메모리(off-chip memory)와의 데이터 교환을 처리하고 상기 신경망 유닛들의 동작을 제어하는 제어기(Controller); 및 신경망의 입력을 제공하는 라인 버퍼(Line Buffer)들, 상기 신경망 유닛들의 동작 과정에서 사용되는 가중치 값들을 저장하는 가중치 버퍼(WeightBuf)들, 그리고 상기 신경망의 출력을 저장하는 출력 버퍼(Output Buffer)를 포함하는 온칩 메모리(on-chip memory);를 포함한다.

대 표 도 - 도3



(52) CPC특허분류
G06F 17/153 (2013.01)
G06N 3/0455 (2023.01)
G06N 3/0464 (2023.01)

(72) 발명자
김지영
서울특별시 광진구 능동로 209, 세종대학교 대양AI센터 712호 (군자동)

한윤호
서울특별시 광진구 능동로 209, 세종대학교 대양AI센터 712호 (군자동)

이 발명을 지원한 국가연구개발사업

과제고유번호	1711179366
과제번호	00156354
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원
연구사업명	정보통신방송혁신인재양성
연구과제명	실-가상 연계 메타버스를 위한 초실감 XR 기술 연구
기 여 율	1/1
과제수행기관명	세종대학교산학협력단
연구기간	2023.01.01 ~ 2023.12.31

명 세 서

청구범위

청구항 1

인코더-디코더 기반의 신경망 구조에 따라 연결되어 신경망 동작을 구현하는 신경망 유닛(Unit)들;

오프칩 메모리(off-chip memory)를 통해 입력 데이터 및 가중치 값들을 독출하고 상기 신경망 동작에 따른 출력 데이터를 저장하고 상기 신경망 유닛들의 파이프라인 동작을 제어하는 제어기(Controller); 및

상기 입력 데이터를 라인 단위로 분할하여 저장하는 라인 버퍼(Line Buffer)들, 상기 신경망 동작의 과정에서 사용되고 각각은 해당 신경망 유닛에 연결되어 해당 가중치 값을 저장하는 가중치 버퍼(WeightBuf)들, 상기 인코더-디코더 간의 스킵 연결을 제공하고 상기 파이프라인 동작을 지원하는 연결 버퍼 그리고 상기 신경망 동작에 따른 출력 데이터를 저장하는 출력 버퍼(Output Buffer)를 포함하는 온칩 메모리(on-chip memory);를 포함하는 인공지능 가속 하드웨어 장치.

청구항 2

제1항에 있어서, 상기 신경망 유닛들은

상기 인코더-디코더(encoder-decoder) 기반의 신경망 구조에서 상기 입력 데이터를 인코딩 데이터로 압축하는 인코딩 과정을 처리하는 인코딩(Encoding) 유닛들;

상기 인코딩 데이터를 복원하는 디코딩 과정을 처리하는 디코딩(Decoding) 유닛들; 및

상기 인코딩 유닛들과 디코딩 유닛들 사이를 연결하는 보틀넥(Bottleneck) 유닛을 포함하는 것을 특징으로 하는 인공지능 가속 하드웨어 장치.

청구항 3

제2항에 있어서, 상기 신경망 유닛들 각각은

상기 입력 데이터에 대한 쉬프트(shift) 연산을 수행하는 쉬프터(Shifter) 모듈;

해당 가중치 버퍼(WeightBuf)로부터 컨볼루션 연산에 필요한 가중치를 수신하는 가중치 레지스터(WeightReg) 모듈;

상기 컨볼루션 연산 과정에서 발생하는 중간 피처맵(intermediate feature map)을 저장하는 임시버퍼(TempBuf) 모듈; 및

상기 컨볼루션 연산을 수행하는 적어도 하나의 컨볼루션 레이어를 구현하는 컨볼루션 유닛(Convolution Unit) 모듈;을 포함하는 것을 특징으로 하는 인공지능 가속 하드웨어 장치.

청구항 4

제3항에 있어서, 상기 인코딩 유닛들 각각은

상기 컨볼루션 유닛 모듈에 의해 수행된 컨볼루션 연산 결과에 대해 다운샘플링(down sampling)을 수행하는 풀버퍼(PoolBuffer) 모듈;을 더 포함하는 것을 특징으로 하는 실시간 인공지능 가속 하드웨어 장치.

청구항 5

제3항에 있어서, 상기 컨볼루션 유닛 모듈은

복수의 곱셈기(multiplier)들을 포함하는 적어도 하나의 PE 모듈;

상기 복수의 곱셈기들의 곱 연산 결과를 합산하는 적어도 하나의 가산기(Adder); 및

상기 가산기에 의해 산출된 합산 결과를 저장하는 적어도 하나의 Tsum 모듈;을 포함하는 것을 특징으로 하는 인공지능 가속 하드웨어 장치.

청구항 6

제5항에 있어서, 상기 컨볼루션 유닛 모듈은

상기 컨볼루션 레이어에 설정된 입력 채널 수, 출력 채널 수 및 필터 크기 중 적어도 하나에 따라 상기 곱셈기, PE 모듈, 가산기 및 Tsum 모듈의 개수를 가변적으로 적용하여 구현되는 것을 특징으로 하는 인공지능 가속 하드웨어 장치.

청구항 7

제1항에 있어서, 상기 제어기는

상기 신경망의 동작 개시 전 단계에서 상기 오프칩 메모리에 접근하여 상기 신경망의 동작을 위한 입력 데이터와 가중치 값들을 독출하고 상기 신경망의 동작 종료 후 단계에서 상기 오프칩 메모리에 접근하여 상기 신경망의 동작에 따른 출력 데이터를 저장하도록 제어하는 것을 특징으로 하는 인공지능 가속 하드웨어 장치.

청구항 8

제1항에 있어서, 상기 제어기는

상기 라인 버퍼들에 저장된 라인 단위의 버퍼 데이터가 입력 데이터의 가장자리 영역과 중첩되는 경우 해당 버퍼 데이터에 대한 패딩(padding) 연산이 개시되도록 제어하는 것을 특징으로 하는 인공지능 가속 하드웨어 장치.

청구항 9

제1항에 있어서, 상기 라인 버퍼들 각각은

상기 오프칩 메모리에서 독출된 입력 데이터를 라인 단위로 분할하여 저장하고 상기 제어기의 제어 신호에 따라 상기 신경망 유닛들에게 라인 단위의 입력 데이터를 제공하는 것을 특징으로 하는 인공지능 가속 하드웨어 장치.

청구항 10

제1항에 있어서, 상기 가중치 버퍼들 각각은

상기 신경망 유닛들과 독립적으로 연결되고 해당 신경망 유닛의 동작 과정에서 사용되는 모든 가중치 값들을 저장하는 것을 특징으로 하는 인공지능 가속 하드웨어 장치.

청구항 11

인공지능 가속 하드웨어 장치에서 수행되는 인공지능 가속 방법에 있어서,

신경망 유닛(Unit)들 통해 인코더-디코더 기반의 신경망 구조에 따라 연결되어 신경망 동작을 구현하는 단계;

제어기(Controller)를 통해 오프칩 메모리(off-chip memory)를 통해 입력 데이터 및 가중치 값들을 독출하고 상기 신경망 동작에 따른 출력 데이터를 저장하고 상기 신경망 유닛들의 파이프라인 동작을 제어하는 단계;

온칩 메모리(on-chip memory)의 라인 버퍼(Line Buffer)들을 통해 상기 입력 데이터를 라인 단위로 분할하여 저장하는 단계;

상기 온칩 메모리(on-chip memory)의 가중치 버퍼(WeightBuf)들을 통해, 상기 신경망 동작의 과정에서 사용되고 각각은 해당 신경망 유닛에 연결되어 해당 가중치 값을 저장하는 단계;

상기 온칩 메모리(on-chip memory)의 연결 버퍼를 통해 상기 인코더-디코더 간의 스킵 연결을 제공하고 상기 파이프라인 동작을 지원하는 단계; 및

상기 온칩 메모리(on-chip memory)의 출력 버퍼(Output Buffer)를 통해 상기 신경망 동작에 따른 출력 데이터를 저장하는 단계를 포함하는 인공지능 가속 방법.

청구항 12

제11항에 있어서, 상기 신경망 동작을 구현하는 단계는

상기 라인 버퍼를 통해 제공된 상기 라인 단위의 버퍼 데이터가 입력 데이터의 가장자리 영역과 중첩되는 경우 해당 버퍼 데이터에 대한 패딩(padding) 연산을 수행하는 단계를 포함하는 것을 특징으로 하는 인공지능 가속 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 신경망 하드웨어 구조에 관한 것으로, 보다 상세하게는 파이프라인화된 신경망 하드웨어 구조를 기반으로 라인 단위의 신경망 연산을 처리하는 각 유닛을 SRAM에 직접 연결함으로써 실시간 노이즈 제거 과정에서 오프칩 메모리에 대한 접근을 최소화하는 메모리 참조에 효과적인 실시간 인공지능 가속 하드웨어 장치 및 그 동작 방법에 관한 것이다.

배 경 기 술

[0003] 현대의 CNN 하드웨어는 고성능과 적은 에너지 소비를 달성하기 위해 발전해 오고 있다. 특히, 오프칩(off-chip) 메모리에 대한 접근이 커질수록 다양한 도전 과제들이 초래될 수 있다. 예를 들어, 레이턴시(Latency)의 경우, Off-chip 메모리 접근은 온칩(on-chip) 메모리 접근에 비해 상당히 느릴 수 있으며, 이로 인해 전체 시스템의 처리 속도가 현저히 저하될 수 있다.

[0004] 또한, Off-chip 메모리 접근은 on-chip 메모리 접근에 비해 더 많은 에너지를 소모할 수 있다. 한편, Off-chip 메모리 컨트롤러는 메모리 접근 요청을 처리하고, 데이터를 읽고 쓰는 역할을 수행할 수 있다. 메모리 컨트롤러는 해당 과정에서 주소 매핑, 데이터 전송, 에러 검출 및 수정 등 다양한 작업을 수행하고 있으며, 이러한 작업들은 메모리 관리의 복잡성을 증가시킬 수 있다.

[0005] 따라서, 전형적인 CNN 하드웨어는 높은 처리량과 적은 에너지 소비를 얻기 위해 병렬 처리, 최적화된 메모리 계층, 그리고 데이터 재사용을 효과적으로 지원하는 기본 설계 원칙을 가질 수 있다. 다만, 카메라 영상 등의 노이즈 제거와 같이 대규모의 연산량과 메모리 참조를 요구하는 실시간성을 달성하기 위해서는 최적화된 하드웨어 구조 설계가 필요할 수 있다.

선행기술문헌

특허문헌

[0007] (특허문헌 0001) 한국공개특허 제10-2023-0117953호 (2023.08.10)

발명의 내용

해결하려는 과제

[0008] 본 발명의 일 실시예는 파이프라인화된 신경망 하드웨어 구조를 기반으로 라인 단위의 신경망 연산을 처리하는 각 유닛을 SRAM에 직접 연결함으로써 실시간 노이즈 제거 과정에서 오프칩 메모리에 대한 접근을 최소화하는 메모리 참조에 효과적인 실시간 인공지능 가속 하드웨어 장치 및 그 동작 방법을 제공하고자 한다.

과제의 해결 수단

[0010] 실시예들 중에서, 메모리 참조에 효과적인 실시간 인공지능 가속 하드웨어 장치는 신경망의 동작을 구현하기 위하여 신경망 구조에 따라 연결된 신경망 유닛(Unit)들; 오프칩 메모리(off-chip memory)와의 데이터 교환을 처리하고 상기 신경망 유닛들의 동작을 제어하는 제어기(Controller); 및 신경망의 입력을 제공하는 라인 버퍼(Line Buffer)들, 상기 신경망 유닛들의 동작 과정에서 사용되는 가중치 값들을 저장하는 가중치 버퍼(WeightBuf)들, 그리고 상기 신경망의 출력을 저장하는 출력 버퍼(Output Buffer)를 포함하는 온칩 메모리(on-chip memory);를 포함한다.

[0011] 상기 신경망 유닛들은 인코더-디코더(encoder-decoder) 구조의 신경망에서 입력 데이터를 인코딩 데이터로 압축하는 인코딩 과정을 처리하는 인코딩 유닛들; 상기 인코딩 데이터를 복원하는 디코딩 과정을 처리하는 디코딩 유닛들; 및 상기 인코딩 유닛들과 디코딩 유닛들 사이를 연결하는 보틀넥(Bottleneck) 유닛을 포함할 수 있다.

[0012] 상기 신경망 유닛들 각각은 입력된 데이터에 대한 쉬프트(shift) 연산을 수행하는 쉬프터(Shifter) 모듈; 상기 가중치 버퍼(WeightBuf)로부터 컨볼루션 연산에 필요한 가중치를 수신하는 가중치 레지스터(WeightReg) 모듈; 상기 컨볼루션 연산 과정에서 발생하는 중간 피처맵(intermediate feature map)을 저장하는 임시버퍼(TempBuf) 모듈; 및 상기 컨볼루션 연산을 수행하는 적어도 하나의 컨볼루션 레이어를 구현하는 컨볼루션 유닛(Convolution Unit) 모듈;을 포함할 수 있다.

[0013] 상기 인코딩 유닛들 각각은 상기 컨볼루션 유닛 모듈에 의해 수행된 컨볼루션 연산 결과에 대해 다운샘플링(down sampling)을 수행하는 풀 버퍼(PoolBuffer) 모듈;을 더 포함할 수 있다.

[0014] 상기 컨볼루션 유닛 모듈은 복수의 곱셈기(multiplier)들을 포함하는 적어도 하나의 PE 모듈; 상기 복수의 곱셈기들의 곱 연산 결과를 합산하는 적어도 하나의 가산기(Adder); 및 상기 가산기에 의해 산출된 합산 결과를 저장하는 적어도 하나의 Tsum 모듈;을 포함할 수 있다.

[0015] 상기 컨볼루션 유닛 모듈은 상기 컨볼루션 레이어에 설정된 입력 채널 수, 출력 채널 수 및 필터 크기 중 적어도 하나에 따라 상기 곱셈기, PE 모듈, 가산기 및 Tsum 모듈의 개수를 가변적으로 적용하여 구현될 수 있다.

[0016] 상기 제어기는 상기 신경망의 동작 개시 전 단계에서 상기 오프칩 메모리에 접근하여 상기 신경망의 동작을 위한 입력 데이터와 가중치 값들을 독출하고 상기 신경망의 동작 종료 후 단계에서 상기 오프칩 메모리에 접근하여 상기 신경망의 동작에 따른 출력 데이터를 저장하도록 제어할 수 있다.

[0017] 상기 제어기는 상기 라인 버퍼들에 저장된 라인 단위의 버퍼 데이터가 입력 데이터의 가장자리 영역과 중첩되는 경우 해당 버퍼 데이터에 대한 패딩(padding) 연산이 개시되도록 제어할 수 있다.

[0018] 상기 라인 버퍼들 각각은 상기 오프칩 메모리에서 독출된 입력 데이터를 라인 단위로 분할하여 저장하고 상기 제어기의 제어 신호에 따라 상기 신경망 유닛들에게 라인 단위의 입력 데이터를 제공할 수 있다.

[0019] 상기 가중치 버퍼들 각각은 상기 신경망 유닛들과 독립적으로 연결되고 해당 신경망 유닛의 동작 과정에서 사용되는 모든 가중치 값들을 저장할 수 있다.

[0020] 실시예들 중에서, 실시간 인공지능 가속 하드웨어 장치에서 수행되는 동작 방법은 오프칩 메모리(off-chip memory)로부터 신경망 유닛들의 동작 과정에서 사용되는 가중치 값들을 독출하여 상기 신경망 유닛들과 독립적

으로 연결된 가중치 버퍼(WeightBuf)들에 저장하는 단계; 라인 버퍼(Line Buffer)들을 통해 상기 오프칩 메모리에서 독출된 신경망의 입력을 상기 신경망 유닛들에 라인 단위로 제공하는 단계; 상기 신경망 유닛들을 통해 상기 신경망 동작을 위한 컨볼루션 연산을 순차적으로 수행하는 단계; 출력 버퍼(Output Buffer)를 통해 상기 컨볼루션 연산의 결과를 상기 신경망의 출력으로서 저장하는 단계; 및 상기 출력 버퍼로부터 상기 신경망의 출력을 독출하여 상기 오프칩 메모리에 저장하는 단계;를 포함한다.

[0021] 상기 컨볼루션 연산을 순차적으로 수행하는 단계는 상기 라인 버퍼를 통해 제공된 상기 라인 단위의 버퍼 데이터가 입력 데이터의 가장자리 영역과 중첩되는 경우 해당 버퍼 데이터에 대한 패딩(padding) 연산을 수행하는 단계를 포함할 수 있다.

발명의 효과

[0023] 개시된 기술은 다음의 효과를 가질 수 있다. 다만, 특정 실시예가 다음의 효과를 전부 포함하여야 한다거나 다음의 효과만을 포함하여야 한다는 의미는 아니므로, 개시된 기술의 권리범위는 이에 의하여 제한되는 것으로 이해되어서는 아니 될 것이다.

[0024] 본 발명의 일 실시예에 따른 메모리 참조에 효과적인 실시간 인공지능 가속 하드웨어 장치 및 그 동작 방법은 파이프라인화된 신경망 하드웨어 구조를 기반으로 라인 단위의 신경망 연산을 처리하는 각 유닛을 SRAM에 직접 연결함으로써 실시간 노이즈 제거 과정에서 오프칩 메모리에 대한 접근을 최소화할 수 있다.

도면의 간단한 설명

[0026] 도 1은 CNN의 컨볼루션 연산의 일 실시예를 설명하는 도면이다.

도 2는 U-net 신경망의 구조를 설명하는 도면이다.

도 3은 CNN 가속 하드웨어 구조를 설명하는 도면이다.

도 4는 Weight Stationary dataflow를 설명하는 도면이다.

도 5는 본 발명에 따른 신경망 하드웨어 구조를 설명하는 도면이다.

도 6은 도 5의 각 신경망 유닛에서의 동작 과정을 설명하는 도면이다.

도 7은 도 5의 각 신경망 유닛의 블록 다이어그램을 설명하는 도면이다.

도 8은 본 발명에 따른 인코딩 유닛의 하드웨어 구조에 관한 블록 다이어그램을 설명하는 도면이다.

도 9는 본 발명에 따른 디코딩 유닛의 하드웨어 구조에 관한 블록 다이어그램을 설명하는 도면이다.

도 10은 본 발명에 따른 컨볼루션 유닛 모듈의 기본 구조를 설명하는 도면이다.

도 11 및 12는 본 발명에 따른 컨볼루션 유닛 모듈의 다양한 실시예를 설명하는 도면이다.

도 13은 본 발명에 따른 패딩 처리 과정을 설명하는 도면이다.

도 14는 본 발명에 따른 라인 방식 입력에 대한 패딩 처리 과정을 설명하는 도면이다.

도 15는 본 발명에 따른 실시간 인공지능 가속 하드웨어 장치의 실시예를 설명하는 도면이다.

발명을 실시하기 위한 구체적인 내용

[0027] 본 발명에 관한 설명은 구조적 내지 기능적 설명을 위한 실시예에 불과하므로, 본 발명의 권리범위는 본문에 설명된 실시예에 의하여 제한되는 것으로 해석되어서는 아니 된다. 즉, 실시예는 다양한 변경이 가능하고 여러 가지 형태를 가질 수 있으므로 본 발명의 권리범위는 기술적 사상을 실현할 수 있는 균등물들을 포함하는 것으로 이해되어야 한다. 또한, 본 발명에서 제시된 목적 또는 효과는 특정 실시예가 이를 전부 포함하여야 한다거나 그러한 효과만을 포함하여야 한다는 의미는 아니므로, 본 발명의 권리범위는 이에 의하여 제한되는 것으로 이해되어서는 아니 될 것이다.

[0028] 한편, 본 출원에서 서술되는 용어의 의미는 다음과 같이 이해되어야 할 것이다.

[0029] "제1", "제2" 등의 용어는 하나의 구성요소를 다른 구성요소로부터 구별하기 위한 것으로, 이들 용어들에 의해 권리범위가 한정되어서는 아니 된다. 예를 들어, 제1 구성요소는 제2 구성요소로 명명될 수 있고, 유사하게 제2 구성요소도 제1 구성요소로 명명될 수 있다.

[0030] 어떤 구성요소가 다른 구성요소에 "연결되어"있다고 언급된 때에는, 그 다른 구성요소에 직접적으로 연결될 수도 있지만, 중간에 다른 구성요소가 존재할 수도 있다고 이해되어야 할 것이다. 반면에, 어떤 구성요소가 다른 구성요소에 "직접 연결되어"있다고 언급된 때에는 중간에 다른 구성요소가 존재하지 않는 것으로 이해되어야 할 것이다. 한편, 구성요소들 간의 관계를 설명하는 다른 표현들, 즉 "~사이에"와 "바로 ~사이에" 또는 "~에 이웃하는"과 "~에 직접 이웃하는" 등도 마찬가지로 해석되어야 한다.

[0031] 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한 복수의 표현을 포함하는 것으로 이해되어야 하고, "포함하다"또는 "가지다" 등의 용어는 실시된 특징, 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것이 존재함을 지정하려는 것이며, 하나 또는 그 이상의 다른 특징이나 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

[0032] 각 단계들에 있어 식별부호(예를 들어, a, b, c 등)는 설명의 편의를 위하여 사용되는 것으로 식별부호는 각 단계들의 순서를 설명하는 것이 아니며, 각 단계들은 문맥상 명백하게 특정 순서를 기재하지 않는 이상 명기된 순서와 다르게 일어날 수 있다. 즉, 각 단계들은 명기된 순서와 동일하게 일어날 수도 있고 실질적으로 동시에 수행될 수도 있으며 반대의 순서대로 수행될 수도 있다.

[0033] 본 발명은 컴퓨터가 읽을 수 있는 기록매체에 컴퓨터가 읽을 수 있는 코드로서 구현될 수 있고, 컴퓨터가 읽을 수 있는 기록 매체는 컴퓨터 시스템에 의하여 읽혀질 수 있는 데이터가 저장되는 모든 종류의 기록 장치를 포함한다. 컴퓨터가 읽을 수 있는 기록 매체의 예로는 ROM, RAM, CD-ROM, 자기 테이프, 플로피 디스크, 광 데이터 저장 장치 등이 있다. 또한, 컴퓨터가 읽을 수 있는 기록 매체는 네트워크로 연결된 컴퓨터 시스템에 분산되어, 분산 방식으로 컴퓨터가 읽을 수 있는 코드가 저장되고 실행될 수 있다.

[0034] 여기서 사용되는 모든 용어들은 다르게 정의되지 않는 한, 본 발명이 속하는 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가진다. 일반적으로 사용되는 사전에 정의되어 있는 용어들은 관련 기술의 문맥상 가지는 의미와 일치하는 것으로 해석되어야 하며, 본 출원에서 명백하게 정의하지 않는 한 이상적이거나 과도하게 형식적인 의미를 지니는 것으로 해석될 수 없다.

[0037] *도 1은 CNN의 컨볼루션 연산의 일 실시예를 설명하는 도면이다.

[0038] 도 1을 참조하면, CNN(Convolutional Neural Network)은 이미지 처리와 관련된 머신 러닝 작업에서 가장 널리 사용되는 신경망에 해당할 수 있다. CNN은 컨볼루션(Convolution) 연산, ReLU(Rectified Linear Unit), 풀링(Pooling) 등의 연산을 포함할 수 있으며, 이 중에서도 컨볼루션 연산은 CNN의 핵심적인 부분에 해당할 수 있다. 즉, 컨볼루션 연산은 필터(또는 커널)라는 작은 행렬을 입력 이미지의 일부분에 겹치게 하여, 매칭되는 원소별 곱(element-wise product)을 계산하고, 그 결과를 모두 더하는 과정으로 진행될 수 있다. 이때, 계산된 값은 새로운 이미지(또는 특성 맵, Feature map)의 픽셀 값이 될 수 있다.

[0039] 또한, 컨볼루션 연산은 필터를 이미지의 다른 부분으로 이동시키면서 전체 이미지에 대해 반복적으로 수행될 수 잇다. 그 결과로 생성된 특성 맵은 원본 이미지의 특정 특징(예를 들어, 가장자리, 색상, 질감 등)을 강조하여 표현될 수 있다. 또한, 필터의 값은 CNN의 학습 과정 중에 생성될 수 있다.

[0040] 구체적으로, 도 1의 그림 (a), (b) 및 (c)는 각각 컨볼루션 연산의 입력, 필터 및 결과를 나타낼 수 있다. 예를 들어, 원본 입력 이미지는 그림 (a)의 파란색 테두리에 해당할 수 있으며, 그림 (a)는 패딩(화살표)처리 된 이미지를 나타낼 수 있다. 또한, 그림 (a)의 음영처리 부분은 필터가 적용된 영역에 해당할 수 있으며, 그림 (c)의 음영처리 부분은 그림 (a)의 음영처리 부분과 그림 (b)에 대한 컨볼루션 연산의 결과에 해당할 수 있다. 2D 컨볼루션 연산은 입력 이미지 위를 필터가 이동하면서 같은 위치의 픽셀끼리의 원소별(element wise) 곱셈의 합을 계산함으로써 수행될 수 있다.

[0042] 도 2는 U-net 신경망의 구조를 설명하는 도면이다.

[0043] 도 2를 참조하면, U-net은 크게 두 부분으로 구성될 수 있다. 하나는 이미지에서 특징을 추출하는 '축소 경로(Contracting path)' 또는 '인코더(Encoding path)'에 해당할 수 있고, 다른 하나는 추출된 특징을 바탕으로 원본 이미지와 같은 해상도의 출력을 생성하는 '확장 경로(Expanding path)' 또는 '디코더(Decoding path)'에 해당할 수 있다.

[0044] 인코더는 컨텍스트(context) 포착을 목적으로 다운샘플링을 수행할 수 있으며, 일반적인 CNN과 유사하게 여러 개의 컨볼루션 레이어와 맥스 풀링(Max Pooling) 레이어를 포함할 수 있다. 도 2에서, 인코더는 4개의 인코딩 블록(Encoding block) (En 0, En 1, ..., En 3)으로 구성될 수 있고, 1개의 인코딩 블록은 3개의 CNN 레이어와 1번의 다운샘플링으로 구성될 수 있다.

[0045] 또한, 디코더는 세밀한 지역화(localization)를 목적으로 업샘플링을 수행할 수 있으며, 인코더와 대칭적으로 여러 개의 업샘플링 레이어와 CNN 레이어를 포함할 수 있다. 도 2에서, 디코더는 4개의 디코딩 블록(Decoding block) (De 0, De 1, ..., De 3)으로 구성될 수 있고, 1개의 디코딩 블록은 3개의 CNN 레이어와 1번의 업샘플링으로 구성될 수 있다.

[0046] 스킵 연결(Skip-connection)은 인코딩 블록과 대칭적인 디코딩 블록 사이를 연결하는 역할을 수행할 수 있다. 해당 연결을 통해 축소된 경로에서의 단순(coarse) 정보를 디코딩 과정에 통합함으로써 학습 효율을 향상시킬 수 있다.

[0047] 보틀넥(Bottleneck)은 신경망에서 차원이 가장 낮은 레이어들로 구성될 수 있다. 즉, 보틀넥은 마지막 다운샘플링과 첫 번째 업샘플링 사이의 CNN 레이어들로 구성될 수 있다. 따라서, 보틀넥은 고수준의 특징(feature)들을 포함한다는 특징을 가질 수 있다.

[0048] U-Net의 구조는 이미지의 세분화된(fine-grained) 특징(세부적이고, 상세한 수준의 정보나 구조)을 잘 보존하면서도 깊은 네트워크를 통해 고차원 특징을 획득할 수 있어, 픽셀 수준의 세부적인 예측을 가능하게 할 수 있다. U-Net의 강력한 특징 추출 능력과 높은 해상도의 출력 생성 능력은 노이즈가 섞인 이미지에서 원본 이미지를 복원하는 데에 매우 유리할 수 있다.

[0050] 도 3은 CNN 가속 하드웨어 구조를 설명하는 도면이다.

[0051] 도 3을 참조하면, CNN 가속 하드웨어 구조는 오프칩(off-chip) 메모리, 온칩(on-chip) 메모리 및 프로세싱 엘리먼트(Processing Elements, PE)의 계층 구조를 포함할 수 있다. 오프칩 메모리는 가장 큰 용량을 가질 수 있지만 접근 속도가 가장 느린 메모리에 해당할 수 있다. 해당 메모리는 주로 모델의 파라미터(weights)와 입력 데이터를 저장하는 데 사용될 수 있다.

[0052] 온칩 메모리는 오프칩 메모리보다 접근 속도가 빠르지만 용량이 작은 메모리에 해당할 수 있다. 해당 메모리는 주로 CNN의 연산 과정에서 발생하는 중간 특징 맵(intermediate feature map)을 저장하는 데 사용될 수 있다. 프로세싱 엘리먼트는 실제로 CNN의 연산을 수행하는 컴퓨팅 리소스에 해당할 수 있다.

[0053] CNN 가속 하드웨어의 동작 과정은 오프칩 메모리에서 데이터 로딩 과정, 온칩 메모리에서 데이터 분배 과정, 프로세싱 엘리먼트에서 연산 수행 과정 및 온칩 메모리에서 결과 로드 과정으로 수행될 수 있다. 구체적으로, 데이터 로딩 과정은 CNN 연산을 시작할 때 필요한 입력 데이터와 가중치를 off-chip 메모리에서 로딩하는 과정에 해당할 수 있다. 이후, 해당 데이터는 on-chip 메모리로 전송되어 빠른 접근이 가능하게 될 수 있다. 또한, 데이터 분배 과정은 로딩된 데이터가 전역 버퍼(Global Buffer)라고 불리는 on-chip 메모리에 저장된 후 필요에 따라 전역 버퍼에서 각 프로세싱 엘리먼트(PE)로 분배되는 과정에 해당할 수 있다.

[0054] 또한, 연산 수행 과정은 각 PE를 통해 주어진 데이터와 파라미터(weights)를 사용하여 필요한 연산을 수행하고, 연산 결과를 다시 on-chip 메모리에 저장하는 과정에 해당할 수 있으며, 저장된 연산 결과는 다음 연산 단계에서 재사용될 수 있다. 마지막으로, 결과 로드 과정은 모든 연산이 완료된 후 최종 결과가 on-chip 메모리에서 로드되는 과정에 해당할 수 있다.

[0055] 한편, 도 3의 2D PE 어레이 방식은 CNN의 구현을 위해서 타일링 기법이 주로 사용될 수 있다. CNN은 대량의 데이터(예를 들어, 입력 feature map, 필터, 출력 feature map)를 처리할 필요가 있으며, 이와 비교하여 가속기(accelerator) HW의 on-chip 메모리는 크기가 제한적이기 때문에 전체 데이터를 한 번에 저장할 수 없다. 이러

한 문제를 해결하기 위해, 데이터는 작은 '타일'로 분할된 후 처리될 수 있다. 각 타일은 2D PE 어레이에 적절한 크기로 설정될 수 있으며, 한 번에 하나의 타일만 on-chip 메모리에 로드되고 처리될 수 있다.

[0056] 타일 기법을 적용할 때 데이터를 저장 혹은 로드하기 위해서 매번 off-chip 메모리를 이용하는 것은 HW의 속도와 에너지 소비 측면에서 불합리적일 수 있다. 이를 해결하기 위해서 다양한 데이터 재사용 전략들이 사용될 수 있다. 즉, 이러한 전략들은 필터의 가중치들(weights)을 PE 레지스터 파일에 저장하여 재사용하는 weights stationary (WS) 방법, 부분 결과 값(partial sum, psum)을 재사용하는 output stationary (OS) 방법, 그리고 필터와 입력 특징맵(feature map), psum 모두를 재사용하는 row stationary (RS) 방법 등을 포함할 수 있다.

[0058] 도 4는 Weight Stationary dataflow를 설명하는 도면이다.

[0059] 도 4를 참조하면, Weight Stationary(WS) dataflow 기법은 CNN 하드웨어 구현에서 사용되는 데이터 플로우 전략 중 하나에 해당할 수 있으며, 가중치(Weight) 데이터를 고정시킨 상태에서 다른 데이터를 이동시키는 방식으로 수행될 수 있다.

[0060] 먼저, 각 PE에는 가중치(Weight)가 로드될 수 있으며, 해당 가중치는 해당 입력들의 연산이 끝날 때까지 고정될 수 있다. 도 4에서는 이러한 가중치를 각각 W0, W1, W2 및 W3으로 나타낼 수 있다. 입력 특징맵(feature map)은 도 4의 Activation으로서 PE 어레이(array)로 스트리밍될 수 있으며, 이는 연속적인 activation이 PE에서 순차적으로 연산에 사용된다는 것을 의미할 수 있다.

[0061] 각 PE는 부분 합(Psum)을 계산하고, 이전의 부분 합(Psum n-1)은 on-chip 메모리에서 독출되어 다른 Psum들과 누적 합산될 수 있다. 이렇게 합쳐진 부분 합(Psum n)은 다시 on-chip 메모리로 전송될 수 있다. 하나의 세트(set)의 가중치에 대한 activation들의 연산이 끝나면, 다음 세트의 가중치가 PE로 로드되어 고정되고 동일한 과정이 반복 수행될 수 있다.

[0062] WS의 주요 특징은 가중치 데이터의 경우 PE 내의 로컬 메모리에 고정되어 여러 입력 데이터와 함께 재사용될 수 있다는 점(재사용성)에 있으며, 가중치 데이터를 고정시키는 대신 입력 데이터와 중간 특징맵(intermediate feature map)의 이동이 빈번하게 발생하는 단점을 가질 수 있다.

[0064] 도 5는 본 발명에 따른 신경망 하드웨어 구조를 설명하는 도면이다.

[0065] 도 5를 참조하면, 실시간 인공지능 가속 하드웨어 장치(500)는 실시간으로 노이즈 제거 동작을 처리하는 신경망 HW 구조로 구현될 수 있으며, 이를 통해 오프칩 메모리에 해당하는 DRAM 접근을 최소화할 수 있다. 이를 위하여, 실시간 인공지능 가속 하드웨어 장치(500)는 '파이프라인화된 신경망 구조', 'Unit과 SRAM의 직접 연결', 'Skip-connection을 위한 버퍼' 및 '라인 단위의 신경망 연산' 등의 디자인 결정(design decision) 요소들을 포함할 수 있다.

[0066] 일 실시예에서, 실시간 인공지능 가속 하드웨어 장치(500)는 사용자 단말과 연동하여 동작하도록 구현될 수 있다. 여기에서, 사용자 단말은 사용자에 의해 운용되는 단말 장치에 해당할 수 있다. 특히, 사용자 단말은 다양한 인공지능 연산을 활용하는 서비스 모듈을 포함할 수 있다. 즉, 사용자 단말은 본 발명에 따른 인공지능 가속 하드웨어 방법을 수행하여 인공지능 모델을 구축하고 이를 활용하기 위한 장치에 해당할 수 있다. 예를 들어, 사용자 단말은 카메라 영상을 촬영하고 카메라 영상에 포함된 노이즈 제거를 위하여 인공지능 모델의 구축을 요청할 수 있다.

[0067] 또한, 사용자 단말은 실시간 인공지능 가속 하드웨어 장치(500)와 연결되어 동작 가능한 스마트폰, 노트북 또는 컴퓨터로 구현될 수 있으며, 반드시 이에 한정되지 않고, 태블릿 PC 등을 포함하여 다양한 디바이스로도 구현될 수 있다. 특히, 사용자 단말은 실시간 인공지능 가속 하드웨어 장치(500)와 연동하기 위한 전용 프로그램 또는 어플리케이션을 설치하여 실행할 수 있다. 또한, 사용자 단말은 실시간 인공지능 가속 하드웨어 시스템을 구성하는 하나의 장치로서 실시간 인공지능 가속 하드웨어 장치(500)와 연동하여 동작할 수 있다.

[0068] 또한, 실시간 인공지능 가속 하드웨어 장치(500)는 독립된 외부 시스템과 연결되어 동작하도록 구현될 수 있다. 예를 들어, 실시간 인공지능 가속 하드웨어 장치(500)는 데이터 전처리를 제공하는 전처리 시스템, 인공지능 모델을 정의하고 관리하는 인공지능 시스템 등과 연동하여 동작할 수 있다.

[0069] 구체적으로, 실시간 인공지능 가속 하드웨어 장치(500)는 신경망을 구성하는 각 레이어들에 대해 인코딩 유닛

(Encoding Unit)(531), 디코딩 유닛(Decoding Unit)(535) 및 보틀넥 유닛(Bottleneck Unit)(533) 등의 유닛 단위로 파이프라인화될 수 있다. 이때, 각 유닛은 신경망의 동작을 구현하기 위하여 신경망 구조에 따라 연결된 신경망 유닛에 해당할 수 있으며, 독립적인 연산을 수행할 수 있다는 점에서 병렬 처리에 적합할 수 있다. 또한, 각 유닛에 포함된 레이어들의 상세 스펙에 맞춰 PE가 최적화되어 배치되기 때문에 필요한 만큼의 PE만을 사용할 수 있어 불필요한 자원 사용을 줄일 수 있다.

[0070] 또한, 실시간 인공지능 가속 하드웨어 장치(500)는 각 유닛이 해당 유닛에서 사용되는 파라미터들, 즉 가중치 값들을 모두 저장하는 on-chip 메모리와 직접 연결되는 방식으로 구현될 수 있다. 또한, 각 유닛 내에서 발생하는 중간 특징맵(intermediate feature map)들도 컨볼루션 유닛(Convolution Unit) 모듈과 직접 연결된 임시버퍼(TempBuffer)에 저장 및 로드될 수 있다. 따라서, 실시간 인공지능 가속 하드웨어 장치(500)는 이미지 입력 및 파라미터(weight)를 처음 로드하는 경우를 제외하고는 off-chip 메모리에 접근하지 않을 수 있다. 이와 달리, 기존의 방식은 모든 레이어가 순차적으로 가속되는 방식을 사용하고 고정된 크기의 2D PE 어레이를 사용하는 점에서 다른 스펙을 가진 다양한 레이어들을 각각 2D PE 어레이 크기에 맞춰 타일링 기법을 적용한 결과 복잡도가 높다는 문제점을 가질 수 있다.

[0071] 또한, 실시간 인공지능 가속 하드웨어 장치(500)는 스킵 연결(skip-connection)을 위한 연결 특징맵(concatenation feature map)들을 저장하기 위해 온칩(on-chip) 메모리만을 사용할 수 있다. 온칩 메모리는 접근 시간이 빠르고 미리 정해진 데이터를 로드하기 때문에 타이밍을 정확하게 제어할 수 있다. 이와 달리, 기존의 방식은 연결 특징맵을 DRAM에 저장하고 필요할 때 로드하는 방식을 사용한 결과 에너지 소비와 시간 지연이 증가한다는 문제점을 가질 수 있다.

[0072] 또한, 실시간 인공지능 가속 하드웨어 장치(500)는 패딩(padding) 처리의 간편함을 위해 전통적인 블록 단위 분할 방법 대신 라인 단위 분할 방법을 적용할 수 있다. 패딩은 엣지에서 처리되는 연산에 해당할 수 있으며, 라인 단위로 입력을 받으면 패딩 처리의 경우의 수가 블록 단위로 입력을 받았을 때보다 적어져 처리가 더 간편해질 수 있다.

[0073] 도 5에서, 실시간 인공지능 가속 하드웨어 장치(500)는 신경망의 이미지 입력(Input)과 파라미터들(Weights)을 로딩할 때, 그리고 최종 결과물을 출력할 때를 제외하고는 외부 접근이 발생하지 않는 하드웨어 구조로 구현될 수 있다. 예를 들어, 실시간 인공지능 가속 하드웨어 장치(500)는 인코더-디코더(encoder-decoder) 구조의 신경망을 구현하는 유닛들을 포함하여 구현될 수 있다.

[0074] 보다 구체적으로, 하드웨어 구조는 인코딩 유닛들(EN 0, EN1)(531), 디코딩 유닛들(DE 0, DE 1)(535), 보틀넥(Bottleneck)(533) 유닛, FC 유닛(537) 및 다수의 버퍼(buffer)들로 구성될 수 있다. 또한, 하드웨어 구조는 오프칩 메모리와의 데이터 교환을 처리하고 각 신경망 유닛들의 동작을 제어하는 제어기(Controller)(510)를 포함할 수 있다. 인코딩 유닛들(531)은 입력 데이터를 인코딩 데이터로 압축하는 인코딩 과정을 처리할 수 있고, 디코딩 유닛들(535)은 인코딩 데이터를 복원하는 디코딩 과정을 처리할 수 있으며, 보틀넥 유닛(533)은 인코딩 유닛들(531)과 디코딩 유닛들(535) 사이를 연결할 수 있다.

[0075] 이때, 버퍼들에는 신경망에 입력을 제공하는 라인 버퍼(Line Buffer)(551)들과, 신경망 유닛들의 동작 과정에서 사용되는 가중치 값들을 저장하는 가중치 버퍼(WeightBuf)(553)들, 신경망의 출력을 저장하는 출력 버퍼(Output Buffer)(555), 그리고 스킵 연결(skip-connection)에 필요한 특징맵(feature map)들을 저장하기 위한 연결 버퍼(CatBuf)(557)들이 포함될 수 있다. 또한, 이러한 버퍼들은 오프칩 메모리에 대한 접근 없이 처리 가능하도록 온칩 메모리 상에서 구현될 수 있다. 예를 들어, 각 버퍼들은 하나의 온칩 메모리 상에서 독립된 영역으로 구분되어 논리적으로 정의되거나 또는 물리적으로 구분된 복수의 온칩 메모리들로 구현될 수 있다.

[0076] 일 실시예에서, 라인 버퍼(551)들 각각은 오프칩 메모리에서 독출된 입력 데이터를 라인 단위로 분할하여 저장하고 제어기(510)의 제어 신호에 따라 신경망 유닛들에게 라인 단위의 입력 데이터를 제공할 수 있다.

[0077] 또한, 각 신경망 유닛에는 가중치 버퍼(553)가 직접 연결될 수 있다. 가중치 버퍼(553)는 각 신경망 유닛들과 독립적으로 연결될 수 있고, 해당 신경망 유닛의 동작 과정에서 수행되는 CNN 연산에 필요한 모든 가중치 값들을 저장할 수 있다. 즉, 가중치들은 입력 데이터와 함께 외부 메모리(즉, off-chip memory)로부터 로드된 후 신경망 구동을 위해 사용될 수 있다.

[0078] 일 실시예에서, 온칩 메모리 기반의 스킵 연결(skip-connection)을 위하여 연결 특징맵(concatenation feature map)을 저장하기 위한 연결 버퍼(CatBuf)(557)들이 인코딩 유닛(531)과 디코딩 유닛(535) 사이에 배치될 수 있다. 이때, 연결 버퍼(557)의 개수는 파이프라인화된 구조에 따라 결정될 수 있으며, 파이프라인에 최적화된 연

결 버퍼(557)의 개수는 신경망의 타이밍을 정확하게 조절하는 데 중요한 역할을 담당할 수 있다.

[0079] 예를 들어, 파이프라인 처리를 위해 EN 0 Unit과 DE 0 Unit 사이에는 3개의 연결 버퍼(CatBuf)들(557)이 필요할 수 있다. 이와 달리, EN 1 Unit과 DE 1 Unit 사이에는 1개의 연결 버퍼(557)만으로 충분할 수 있다. 만약 추가적인 인코딩 유닛(531)이 도입된다면 디코딩 유닛(535)도 동일하게 추가되어야 하며, 이에 대한 파이프라인 처리의 타이밍을 맞추기 위해서는 2개의 추가 연결 버퍼(557)들이 필요할 수 있다.

[0080] 일 실시예에서, 제어기(510)는 신경망의 동작 개시 전 단계에서 오프칩 메모리에 접근하여 신경망의 동작을 위한 입력 데이터와 가중치 값들을 독출하고 신경망의 동작 종료 후 단계에서 오프칩 메모리에 접근하여 신경망의 동작에 따른 출력 데이터를 저장하도록 제어할 수 있다. 다른 실시예에서, 제어기(510)는 라인 버퍼(551)들에 저장된 라인 단위의 버퍼 데이터가 입력 데이터의 가장자리 영역과 중첩되는 경우 해당 버퍼 데이터에 대한 패딩(padding) 연산이 개시되도록 제어할 수 있다.

[0082] 도 6은 도 5의 각 신경망 유닛에서의 동작 과정을 설명하는 도면이다.

[0083] 도 6을 참조하면, 그림 (a)는 인코딩 유닛(Encoding Unit)에 대한 동작 과정에 해당하고, 그림 (b) 및 (c)는 각각 보틀넥 유닛(Bottleneck Unit) 및 디코딩 유닛(Decoding Unit)에 대한 동작 과정에 해당할 수 있다.

[0084] 구체적으로, 신경망에서 사용되는 모든 가중치들은 해당 유닛과 연결된 가중치 버퍼(WeightBuf)로 전송될 수 있다. 예를 들면, EN 0 Unit에서 사용되는 가중치들은 EN 0 Unit에 연결된 가중치 버퍼로 전송될 수 있다. 가중치들이 모두 로드되면, 인코딩 유닛은 라인 버퍼(Line Buffer)를 통해 입력 특징맵(feature map)을 수신할 수 있다. 입력 특징맵이 전달되면, 가중치 버퍼에서 가중치들이 인코딩 유닛으로 전달되고 합성 곱 연산이 개시될 수 있다. 이때, 해당 연산은 유닛 내의 레이어 순서에 따라 차례대로 수행될 수 있다.

[0085] 또한, 연산 결과가 연결 특징맵(concatenation feature map)인 경우, 연결 버퍼(CatBuf)에 저장될 수 있다. 만약 그렇지 않다면, 해당 연산이 마지막 레이어에 대한 것인지를 확인하고, 마지막이라면 인코딩 유닛의 작업은 종료될 수 있다. 그렇지 않다면, 라인 버퍼를 통해 새로운 입력이 수신되고 동일한 과정이 계속 반복 수행될 수 있다.

[0086] 보틀넥 유닛은 마지막 인코딩 유닛의 출력을 입력으로 수신할 수 있다. 입력 특징맵의 전달이 완료되면, 가중치 버퍼에서 가중치들이 보틀넥 유닛으로 전달되고 합성 곱 연산이 개시될 수 있다. 이러한 연산은 유닛 내의 레이어 순서에 따라 차례로 수행될 수 있다. 해당 연산이 마지막 레이어에 대한 것인지를 확인하고, 마지막이라면 보틀넥 유닛의 작업은 종료될 수 있다. 그렇지 않다면, 새로운 입력이 수신되고 동일한 과정이 계속 반복 수행될 수 있다.

[0087] 디코딩 유닛은 보틀넥 유닛의 출력을 입력으로 수신할 수 있다. 디코딩 유닛은 연결 특징맵도 입력으로 사용하기 때문에 연결 버퍼로부터 연결 특징맵을 수신할 수 있다. 입력에 대한 전달이 완료되면, 가중치 버퍼에서 가중치들이 디코딩 유닛으로 전달되고 합성 곱 연산이 개시될 수 있다. 이러한 연산은 유닛 내의 레이어 순서에 따라 차례로 수행될 수 있다. 해당 연산이 마지막 레이어에 대한 것인지를 확인하고, 마지막이라면 디코딩 유닛의 작업은 종료될 수 있다. 그렇지 않다면, 새로운 입력이 수신되고 동일한 과정이 계속 반복 수행될 수 있다.

[0089] 도 7은 도 5의 각 신경망 유닛의 블록 다이어그램을 설명하는 도면이고, 도 8은 본 발명에 따른 인코딩 유닛의 하드웨어 구조에 관한 블록 다이어그램을 설명하는 도면이며, 도 9는 본 발명에 따른 디코딩 유닛의 하드웨어 구조에 관한 블록 다이어그램을 설명하는 도면이다.

[0090] 도 7을 참조하면, 실시간 인공지능 가속 하드웨어 장치는 신경망의 동작을 구현하기 위하여 신경망 구조에 따라 연결된 신경망 유닛(Unit)들을 포함할 수 있다. 이때, 신경망 유닛들은 인코딩 유닛(Encoding Unit), 보틀넥 유닛(Bottleneck Unit), 디코딩 유닛(Decoding Unit) 및 FC 유닛(FC Unit)을 포함할 수 있다. 신경망을 구성하는 각 유닛은 컨볼루션(convolution) 연산을 위한 기본 모듈들을 포함하여 구성될 수 있으며, 각 유닛의 역할에 따라 세부 구성이 조금씩 달라질 수 있다. 예를 들어, 다운샘플링을 포함하는 유닛, 잔차 연결(residual connection) 혹은 결합(concatenation)을 포함하는 유닛들은 세부 기능에 따라 상이한 구성으로 구현될 수 있다.

[0091] 각 유닛에 공통적으로 적용되는 기본 구조(Basis Structure)에는 쉬프터(Shifter), 컨볼루션 유닛(Convolution

Unit), 가중치 버퍼(WeightBuf), 임시 버퍼(TempBuffer) 모듈들이 포함될 수 있다. 구체적으로, 쉬프터 모듈에서는 컨볼루션 연산을 위하여 입력 데이터(Input)에 대한 쉬프트(shift) 연산이 수행될 수 있다. 이때, 2D 컨볼루션 연산은 필터들이 픽셀단위로 입력 특징맵을 슬라이딩(sliding) 이동하면서 수행될 수 있다.

[0092] 또한, 가중치 버퍼 모듈은 각 유닛에서 사용하는 모든 가중치들을 호스트(Host)로부터 수신하여 저장할 수 있다. 가중치 레지스터 모듈은 컨볼루션 연산에 필요한 가중치들을 가중치 버퍼 모듈로부터 수신할 수 있다. 임시 버퍼 모듈은 각 유닛의 마지막 컨볼루션 레이어를 제외한 나머지 레이어들에서 발생하는 중간 특징맵(intermediate feature map)을 저장할 수 있다. 컨볼루션 유닛 모듈에서는 각 유닛에 포함된 CNN 레이어들의 실질적인 컨볼루션 연산이 순차적으로 수행될 수 있다. 컨볼루션 유닛 모듈에는 PE들이 레이어 설정(configuration)에 최적화되어 구성될 수 있다.

[0093] 도 8을 참조하면, 각 인코딩 유닛(Encoding Unit)은 다운샘플링을 수행할 수 있으며, 컨볼루션 유닛 모듈에서는 연결 특징맵(concatenation feature map)이 생성될 수 있다. 즉, 각 인코딩 유닛은 도 7의 기본 구조를 바탕으로 다운샘플링을 위한 풀 버퍼(PoolBuffer) 모듈을 포함할 수 있다. 즉, 풀 버퍼 모듈은 컨볼루션 유닛 모듈에 의해 수행된 컨볼루션 연산 결과에 대해 다운샘플링을 수행할 수 있다. 일 실시예에서, 각 인코딩 유닛은 연결 특징맵을 저장하는 연결 버퍼(CatBuf)와 연결되도록 구현될 수 있다.

[0094] 도 9를 참조하면, 각 디코딩 유닛(Decoding Unit)은 연결 특징맵을 수신하기 위하여 연결 특징맵을 저장하는 연결 버퍼(CatBuf)와 연결될 수 있다. 한편, 보틀넥 유닛(Bottleneck Unit)과 FC 유닛(FC Unit)은 컨볼루션 연산만으로 구성될 수 있으며, 이에 따라 도 7의 기본 구조와 동일한 구조로 구현될 수 있다.

[0096] 도 10은 본 발명에 따른 컨볼루션 유닛 모듈의 기본 구조를 설명하는 도면이고, 도 11 및 12는 본 발명에 따른 컨볼루션 유닛 모듈의 다양한 실시예를 설명하는 도면이다.

[0097] 도 10을 참조하면, 각 유닛은 기본적으로 컨볼루션 연산을 위한 컨볼루션 유닛 모듈을 포함하여 구현될 수 있다. 컨볼루션 유닛 모듈은 컨볼루션 레이어의 상세 스펙에 최적화된 PE를 포함하여 구현될 수 있다. 고정된 크기의 2D PE 어레이를 사용하는 기존 방식의 경우, 복잡성을 증가시키고 구현 난이도를 높일 수 있는 레이어들에 대한 타일링이 필요할 수 있으며, 각 레이어의 특성에 따라 일부 PE가 비활성 상태로 존재할 수 있다.

[0098] 일 실시예에서, 컨볼루션 유닛 모듈은 다수 개의 프로세싱 엘리먼트(PE), Adder 및 컨볼루션 연산의 결과인 Tsum으로 구성될 수 있다. 특히, PE의 경우 일반적인 CNN 하드웨어에 포함된 MAC 유닛이 아닌 복수 개의 곱셈기(multiplier)들을 포함하는 곱셈기 그룹으로 구성될 수 있다. 각 PE에서는 가중치(weight)와 활성화(activation) 맵 간의 곱 연산이 수행될 수 있다. 가산기(Adder)는 각 PE의 결과값을 수신하여 전체 합(total sum) 연산을 수행할 수 있으며, 연산 결과를 Tsum에 저장할 수 있다. 즉, 가산기는 PE마다 연결되어 하나의 PE에 포함된 복수의 곱셈기들의 연산 결과를 수신하여 합산하는 동작을 수행할 수 있다. 연산 결과가 저장되는 Tsum은 신경망 구조 상 해당 연산의 위치에 따라 임시 버퍼 또는 출력 버퍼와 연결될 수 있다. 만약 Tsum이 현재 연산 중인 레이어의 중간 특징맵인 경우 임시 버퍼(TempBuffer)에 저장될 수 있으며, 최종 특징맵인 경우 출력 버퍼(Output Buffer)에 저장될 수 있다.

표 1

[0099]

	입력 채널의 수	출력 채널의 수	필터 크기
레이어 1	10	45	3×3
레이어 2	45	60	1×1

[0100] 상기의 표 1의 경우 특정 유닛에서의 레이어 구성의 일 실시예에 해당할 수 있다. 해당 실시예의 경우, 두 개의 연속된 CNN 레이어로 구성될 수 있으며, 필터의 크기는 3×3 및 1×1이 사용될 수 있다. 한편, 필터의 크기는 다양한 크기의 값이 적용될 수 있음은 물론이다.도 11을 참조하면, 표 1의 레이어 1을 처리하는 컨볼루션 유닛 모듈의 일 실시예에 해당할 수 있다. 해당 컨볼루션 유닛 모듈에는 9개의 곱셈기(multiplier)로 구성된 PE가 총 30개 사용될 수 있다. 즉, 레이어 1에서는 3×3 크기의 필터가 사용될 수 있으며, 입력 채널이 10개이기 때문에 하나의 출력 채널을 계산하기 위하여 90개의 곱셈기가 필요할 수 있다. 따라서, 10개의 PE를 통해 하나의 출력 채널을 처리할 수 있다. 도 11의 경우, 30개의 PE를 사용할 수 있기 때문에 3개의 채널(즉, Tsum 0, 1, 2)을 동시에 출력할 수 있다. 레이어 1은 총 45개의 출력 채널을 포함할 수 있으며, 해당 연산을 15번 반복하는 경우

레이어 1의 컨볼루션 연산이 완료될 수 있다. 이를 통해 산출된 결과 값들(즉, Tsum 0, 1, 2, ..., 44)은 레이어 2의 입력으로 사용될 수 있으며, 이를 위하여 임시 버퍼(TempBuffer)에 저장될 수 있다.

[0101] 도 12를 참조하면, 표 1의 레이어 2를 처리하는 컨볼루션 유닛 모듈의 일 실시예에 해당할 수 있다. 레이어 2의 입력(Activation)은 임시 버퍼(TempBuffer)로부터(즉, 레이어 1의 final feature map) 전달될 수 있다. 레이어 2에서는 1×1 크기의 필터가 사용될 수 있으며, 입력 채널이 45개이기 때문에 하나의 출력 채널을 계산하기 위하여 45개의 곱셈기가 필요할 수 있다. 따라서, 5개의 PE를 통해 하나의 출력 채널을 처리할 수 있다. 도 12의 경우, 30개의 PE를 사용할 수 있기 때문에 6개의 채널(즉, Tsum 0, 1, ..., 5)을 동시에 출력할 수 있다. 레이어 2는 총 60개의 출력 채널을 포함할 수 있으며, 해당 연산을 10번 반복하는 경우 레이어 2의 컨볼루션 연산이 완료될 수 있다. 이를 통해 산출된 결과 값들(즉, Tsum 0, 1, 2, ..., 59)은 출력 버퍼(Output Buffer)로 전송될 수 있다.

[0102] 일 실시예에서, 컨볼루션 유닛 모듈은 컨볼루션 레이어에 설정된 입력 채널 수, 출력 채널 수 및 필터 크기 중 적어도 하나에 따라 곱셈기, PE 모듈, 가산기 및 Tsum 모듈의 개수를 가변적으로 적용하여 구현될 수 있다. 즉, 컨볼루션 유닛 모듈은 신경망 레이어들의 구성 및 상세 스펙 그리고 신경망 하드웨어의 목표 성능에 따라 가변적으로 적용될 수 있다. 따라서, PE의 개수, PE 1개에 포함되는 곱셈기의 개수 등과 같은 컨볼루션 유닛 모듈의 구성요소들은 특정 개수로 고정되지 않고 구현 환경에 따라 적응적으로 조정될 수 있다.

[0104] 도 13은 본 발명에 따른 패딩 처리 과정을 설명하는 도면이다.

[0105] 도 13을 참조하면, 이미지 형태의 입력을 전달하는 방법에는 블록 단위의 방식과 라인 단위의 방식을 포함할 수 있다. 일 실시예에서, 실시간 인공지능 가속 하드웨어 장치는 입력에 대한 패딩의 처리가 블록 단위의 입력 방식보다 상대적으로 수월한 라인 단위의 입력 방식이 적용되어 구현될 수 있다.

[0106] 예를 들어, 도 13에서, 왼쪽 그림은 12×12 크기의 입력 데이터에 해당할 수 있다. 또한, 오른쪽 그림은 블록 또는 라인 방식으로 입력을 전달받은 후 패딩 처리되는 과정을 나타낼 수 있다. 이때, 그림의 각 번호들은 블록 또는 라인 방식으로 분할된 입력 데이터를 나타낼 수 있다.

[0107] 보다 구체적으로, 오른쪽 상단 그림에서, 입력 블록 1의 상변과 좌변의 픽셀들은 원본 입력 데이터의 가장자리 영역에 포함될 수 있다. 따라서, 입력 블록 1은 상변과 좌변에 대해 패딩 처리가 필요할 수 있다. 또한, 입력 블록 2, 3 및 4는 각각 상변, 상변과 우변, 그리고 좌변에 대해 패딩 처리가 필요할 수 있다. 입력 블록 5의 경우, 어느 픽셀도 원본 입력 데이터의 가장자리 영역에 포함되지 않기 때문에 패딩 처리가 필요하지 않을 수 있다. 입력 블록 6, 7, 8 및 9는 각각 우변, 하변과 좌변, 하변, 그리고 하변과 우변에 대해 패딩 처리가 필요할 수 있다. 결과적으로, 입력 블록이 원본 입력 데이터의 가장자리 영역과 중첩되는 위치에 따라 블록 방식의 입력은 총 9가지의 패딩 처리 방식이 필요할 수 있다.

[0108] 또한, 오른쪽 하단 그림에서, 입력 라인 1과 같이 입력 라인이 패딩 위치에 존재하는 경우 해당 입력 라인에 대해 패딩 처리가 수행될 수 있다. 또한, 입력 라인 2의 경우 입력 라인의 가장 앞부분에 대해서만 패딩 처리가 수행될 수 있으며, 입력 라인 3의 경우 가장 뒷부분에 대해서만 패딩 처리가 수행될 수 있다. 입력 라인 4의 경우 패딩 처리가 필요하지 않을 수 있다. 따라서, 라인 방식의 입력에 대해서는 총 4가지의 패딩 처리 방식이 필요할 수 있으며, 블록 단위 입력 방식이 요구하는 총 9가지의 패딩 처리 방식에 비해 훨씬 적은 수의 처리 방식만이 필요하다는 점에서 라인 단위 입력 방식은 패딩 처리의 간편함 측면에서 블록 단위 입력 방식보다 우수할 수 있다.

[0110] 도 14는 본 발명에 따른 라인 방식 입력에 대한 패딩 처리 과정을 설명하는 도면이다.

[0111] 도 14를 참조하면, 실시간 인공지능 가속 하드웨어 장치는 라인 단위의 입력 방식을 통해 패딩 처리를 수행할 수 있다. 보다 구체적으로, 실시간 인공지능 가속 하드웨어 장치는 라인 버퍼가 입력 특징맵(input feature map)의 첫 줄을 읽기 전이거나 마지막 줄을 읽은 후인지 여부를 검출할 수 있다.

[0112] 만약 그렇다면, 실시간 인공지능 가속 하드웨어 장치는 라인 버퍼 전체에 대한 패딩 처리를 개시할 수 있다. 만약 그렇지 않다면, 실시간 인공지능 가속 하드웨어 장치는 라인 버퍼가 입력 특징맵의 테두리 영역을 포함하는지 여부를 검출할 수 있다. 만약 포함한다면, 실시간 인공지능 가속 하드웨어 장치는 그것이 앞부분인지 또는 뒷부분인지 여부를 검출할 수 있고, 해당 부분에 대해 패딩 처리를 수행할 수 있다. 만약 테두리를 포함하지 않

는다면, 실시간 인공지능 가속 하드웨어 장치는 해당 부분에 대해 패딩 처리를 수행하지 않을 수 있다.

[0114] 도 15는 본 발명에 따른 실시간 인공지능 가속 하드웨어 장치의 실시예를 설명하는 도면이다.

[0115] 도 15를 참조하면, 실시간 인공지능 가속 하드웨어 장치(1530)는 호스트 PC(1510)와 연동하여 동작하도록 구현될 수 있다. 예를 들어, 그림 (a)는 호스트 PC(1510)와 실시간 인공지능 가속 하드웨어 장치(1530)가 독립적인 장치로 구현된 결과 상호 연동하여 동작하는 경우에 해당할 수 있다. 이 경우, 실시간 인공지능 가속 하드웨어 장치(1530)는 호스트 PC(1510)의 내부 메모리(1511)에 접근하여 신경망 동작을 위한 데이터들을 수신할 수 있고, 신경망 동작에 따른 결과를 저장할 수 있다. 즉, 실시간 인공지능 가속 하드웨어 장치(1530)는 호스트 PC(1510)와 연동하여 오프칩 메모리에 해당하는 내부 메모리(1511)에 접근할 수 있으며, 신경망 동작 과정에서 온칩 메모리(1531)를 활용함으로써 오프칩 메모리에 대한 접근을 최소화할 수 있다.

[0116] 다른 예로서, 그림 (b)는 실시간 인공지능 가속 하드웨어 장치(1530)가 호스트 PC(1510)의 내부에 포함되어 구현되는 경우에 해당할 수 있다. 이 경우, 실시간 인공지능 가속 하드웨어 장치(1530)는 호스트 PC(1510)의 내부 메모리(1511)와 버스(bus)를 통해 연결될 수 있으며, 내부 메모리(1511)는 오프칩 메모리로서 동작할 수 있다.

[0118] 상기에서는 본 발명의 바람직한 실시예를 참조하여 설명하였지만, 해당 기술 분야의 숙련된 당업자는 하기의 특허 청구의 범위에 기재된 본 발명의 사상 및 영역으로부터 벗어나지 않는 범위 내에서 본 발명을 다양하게 수정 및 변경시킬 수 있음을 이해할 수 있을 것이다.

부호의 설명

[0120] 500: 실시간 인공지능 가속 하드웨어 장치

510: 제어기 531: 인코딩 유닛

533: 보틀넥 유닛 535: 디코딩 유닛

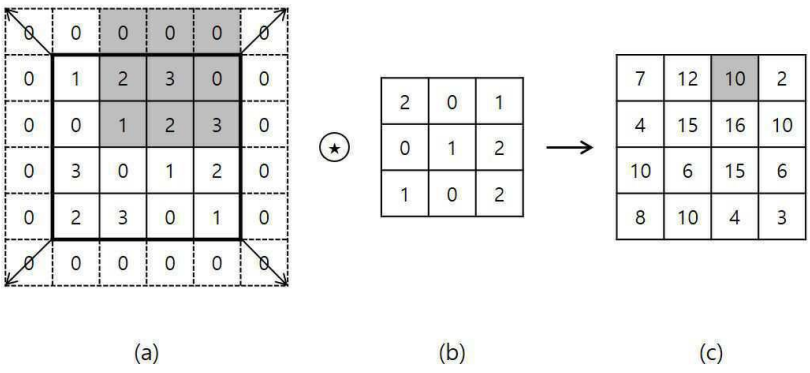
537: FC 유닛

551: 라인 버퍼 553: 가중치 버퍼

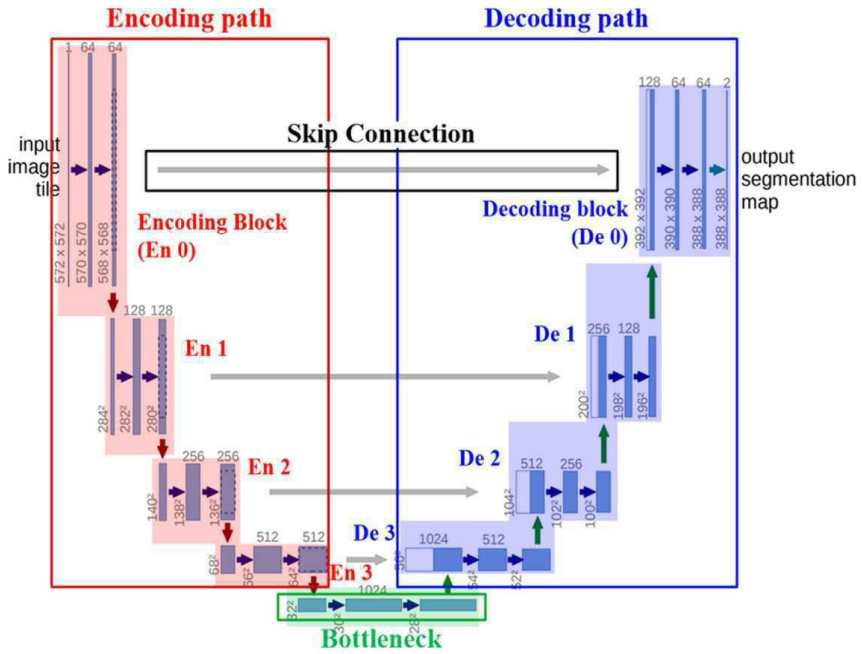
555: 출력 버퍼 557: 연결 버퍼

도면

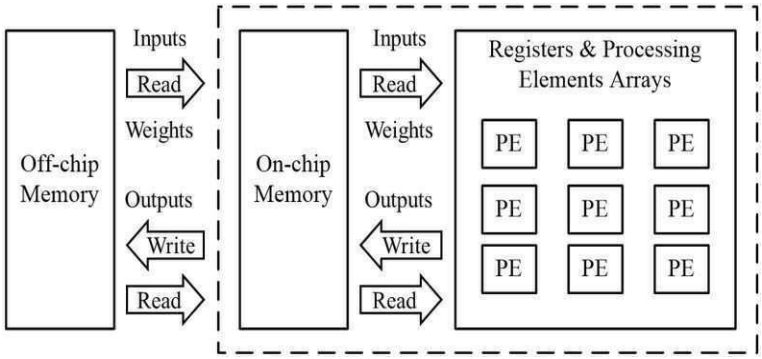
도면1



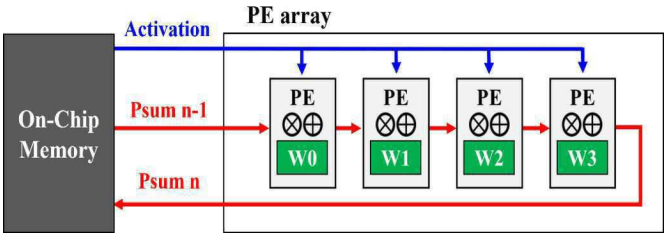
도면2


Encoding path
Decoding path
Skip Connection
input image tile
output segmentation map
Encoding Block (En 0)
Decoding block (De 0)
En 1
En 2
En 3
De 1
De 2
De 3
Bottleneck

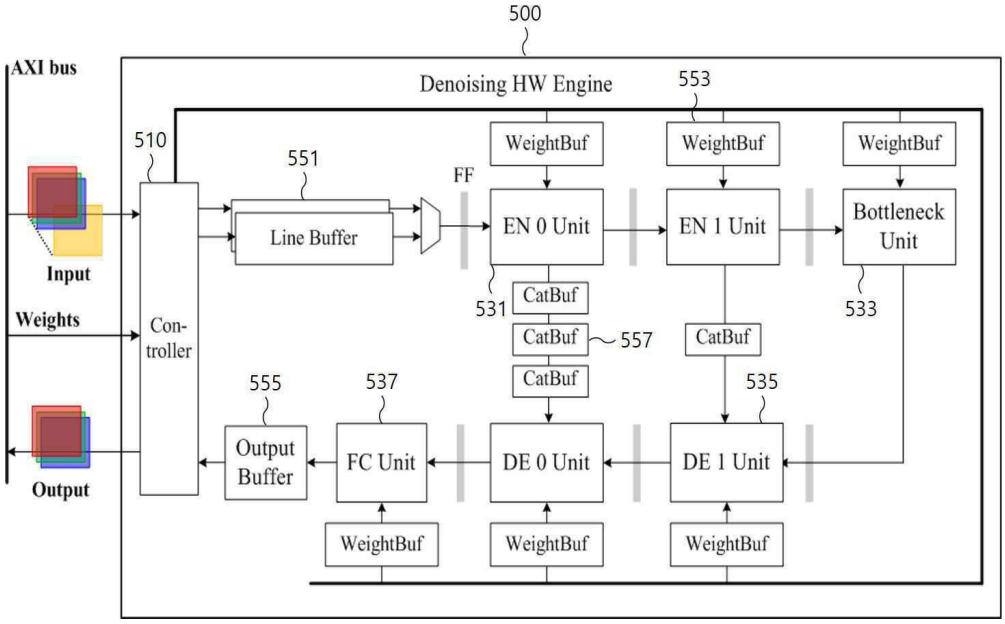
도면3


Off-chip Memory
Inputs
Read
Weights
Outputs
Write
Read
On-chip Memory
Inputs
Read
Weights
Outputs
Write
Read
Registers & Processing Elements Arrays
PE

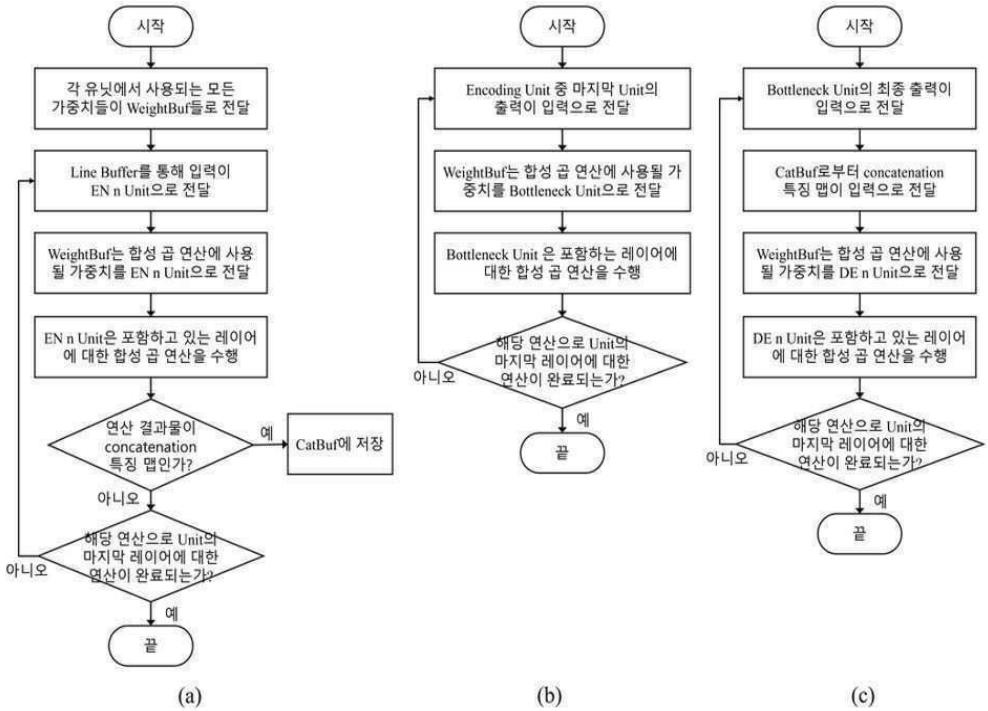
도면4


PE array
Activation
On-Chip Memory
Psum n-1
Psum n
PE
W0
W1
W2
W3

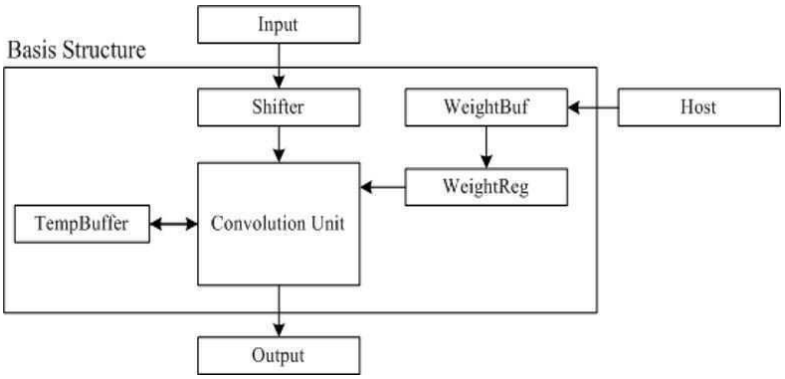
도면5


500
AXI bus
Denoising HW Engine
553
510
551
FF
WeightBuf
WeightBuf
WeightBuf
Line Buffer
EN 0 Unit
EN 1 Unit
Bottleneck Unit
Input
Weights
Con-troller
531
CatBuf
CatBuf
557
CatBuf
CatBuf
533
555
537
535
Output Buffer
FC Unit
DE 0 Unit
DE 1 Unit
Output
WeightBuf
WeightBuf
WeightBuf

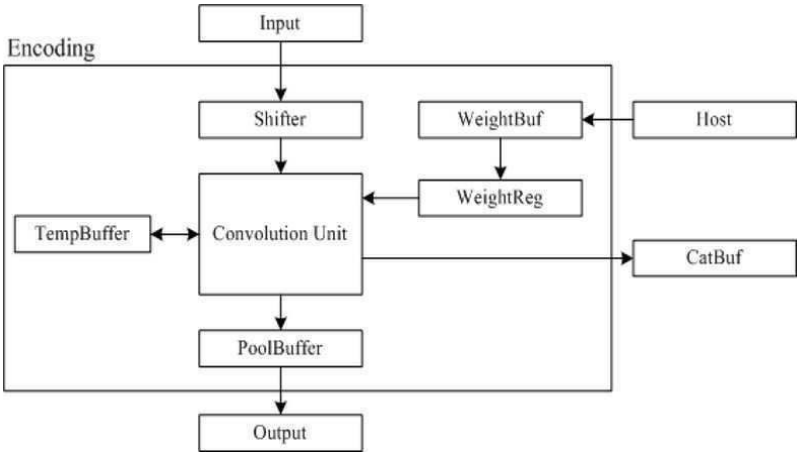
도면6


시작
각 유닛에서 사용되는 모든 가중치들이 WeightBuf들로 전달
Line Buffer를 통해 입력이 EN n Unit으로 전달
WeightBuf는 합성 곱 연산에 사용될 가중치를 EN n Unit으로 전달
EN n Unit은 포함하고 있는 레이어에 대한 합성 곱 연산을 수행
연산 결과물이 concatenation 특징 맵인가?
예
CatBuf에 저장
아니오
해당 연산으로 Unit의 마지막 레이어에 대한 연산이 완료되는가?
아니오
예
끝
(a)
시작
Encoding Unit 중 마지막 Unit의 출력이 입력으로 전달
WeightBuf는 합성 곱 연산에 사용될 가중치를 Bottleneck Unit으로 전달
Bottleneck Unit 은 포함하는 레이어에 대한 합성 곱 연산을 수행
해당 연산으로 Unit의 마지막 레이어에 대한 연산이 완료되는가?
아니오
예
끝
(b)
시작
Bottleneck Unit의 최종 출력이 입력으로 전달
CatBuf로부터 concatenation 특징 맵이 입력으로 전달
WeightBuf는 합성 곱 연산에 사용될 가중치를 DE n Unit으로 전달
DE n Unit은 포함하고 있는 레이어에 대한 합성 곱 연산을 수행
해당 연산으로 Unit의 마지막 레이어에 대한 연산이 완료되는가?
아니오
예
끝
(c)

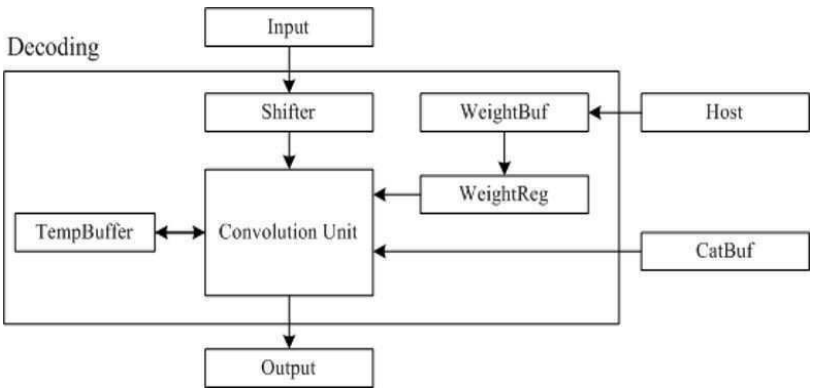
도면7


Basis Structure
Input
Shifter
WeightBuf
Host
WeightReg
TempBuffer
Convolution Unit
Output

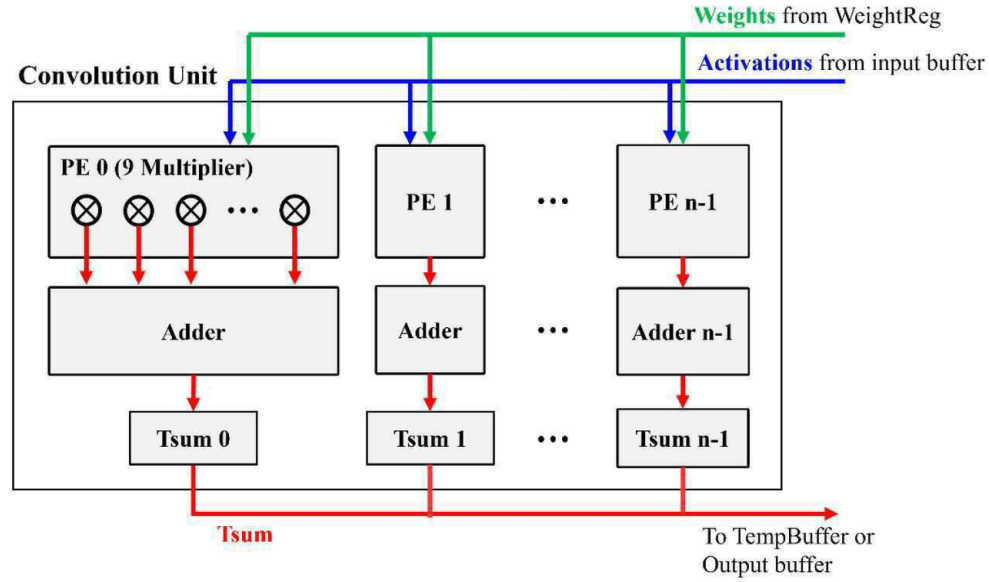
도면8


Encoding
Input
Shifter
WeightBuf
Host
WeightReg
TempBuffer
Convolution Unit
CatBuf
PoolBuffer
Output

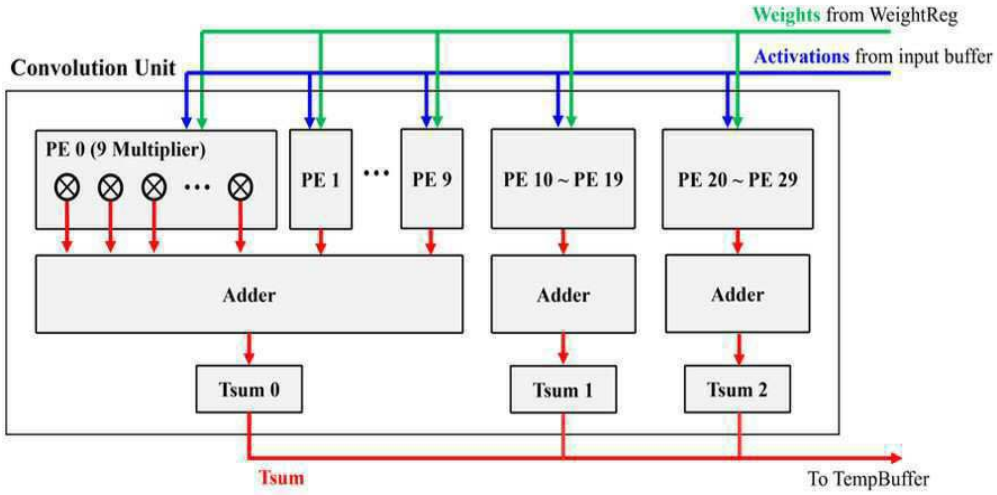
도면9


Decoding
Input
Shifter
WeightBuf
Host
WeightReg
TempBuffer
Convolution Unit
CatBuf
Output

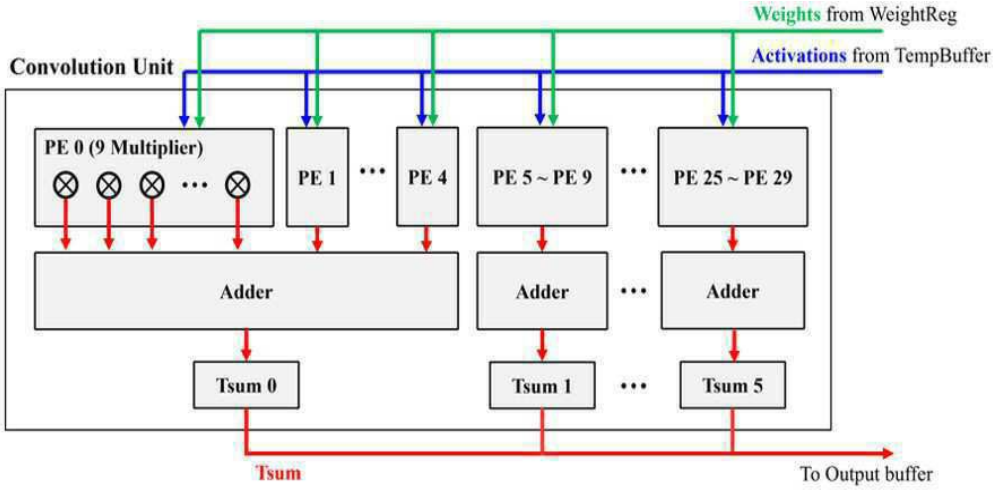
도면10



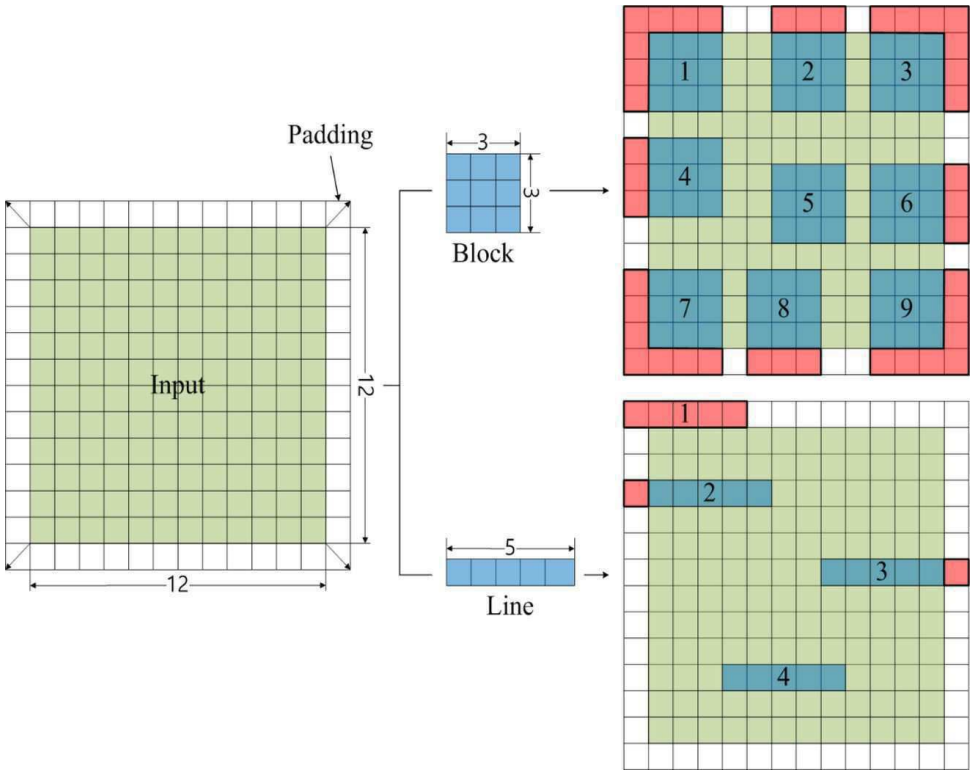
도면11



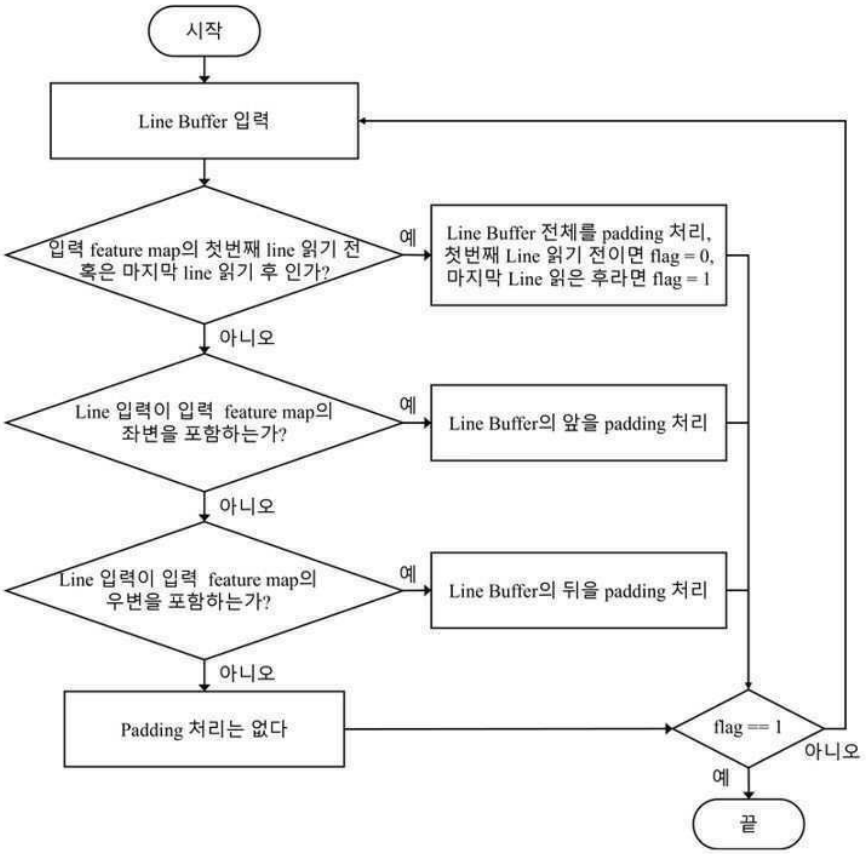
도면12



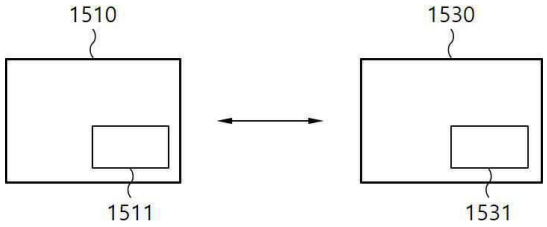
도면13


Padding
3
3
Block
Input
12
12
5
Line
1
2
3
4
5
6
7
8
9
1
2
3
4

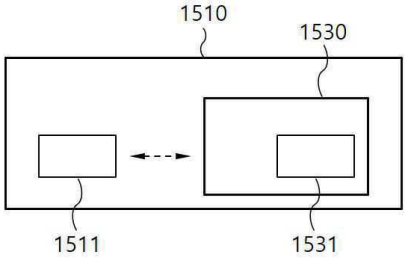
도면14


시작
Line Buffer 입력
입력 feature map의 첫번째 line 읽기 전 혹은 마지막 line 읽기 후 인가?
예
Line Buffer 전체를 padding 처리, 첫번째 Line 읽기 전이면 flag = 0, 마지막 Line 읽은 후라면 flag = 1
아니오
Line 입력이 입력 feature map의 좌변을 포함하는가?
예
Line Buffer의 앞을 padding 처리
아니오
Line 입력이 입력 feature map의 우변을 포함하는가?
예
Line Buffer의 뒤을 padding 처리
아니오
Padding 처리는 없다
flag == 1
아니오
예
끝

도면15



(a)



(b)