(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2024-0161452
(43) 공개일자 2024년11월12일

(51) 국제특허분류(Int. Cl.)
H04L 9/00 (2022.01) *G06N 3/0464* (2023.01)
G06N 3/063 (2023.01)
(52) CPC특허분류
H04L 9/00 (2022.05)
G06N 3/0464 (2023.01)
(21) 출원번호 10-2023-0058605
(22) 출원일자 2023년05월04일
심사청구일자 2023년05월04일

(71) 출원인
연세대학교 산학협력단
서울특별시 서대문구 연세로 50 (신촌동, 연세대학교)
(72) 발명자
김영석
서울시 서대문구 연세로 50
최진우
서울특별시 서대문구 연세로 50
(뒷면에 계속)
(74) 대리인
정부연

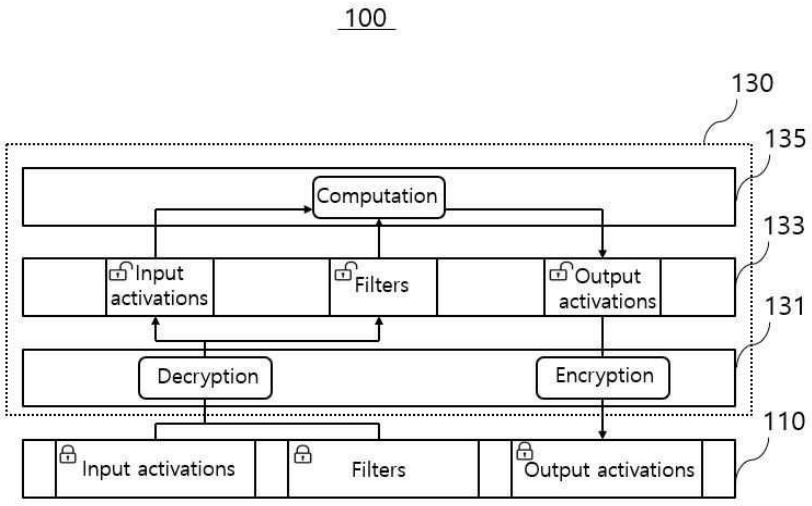
전체 청구항 수 : 총 15 항

(54) 발명의 명칭 **트러스트 환경 기반 인공지능 장치**

(57) 요 약

본 발명은 트러스트 환경 기반 인공지능 장치에 관한 것으로, 암호화 입력 데이터를 송신하고 암호화 출력 데이터를 수신하는 제1 유형의 메모리; 및 트러스트 신뢰 공간에서 동작되고 상기 암호화 입력 및 출력 데이터의 인공지능 연산을 수행하는 트러스트 인공지능 처리부를 포함하고, 상기 트러스트 인공지능 처리부는 상기 암호화 입력 데이터의 복호화를 통해 복호화 입력 데이터를 생성하고 상기 암호화 출력 데이터를 생성하기 위해 비암호화 출력 데이터의 암호화를 수행하는 암호처리 프론트-엔드 프로세서; 상기 복호화 입력 데이터 및 상기 비암호화 출력 데이터에 대한 버퍼를 제공하는 제2 유형의 메모리; 및 상기 복호화 입력 데이터를 기초로 신경망 연산을 수행하여 상기 비암호화 출력 데이터를 생성하는 프로세서를 포함한다.

대 표 도 - 도1



(52) CPC특허분류
G06N 3/063 (2013.01)
H04L 2209/125 (2013.01)
H04L 2209/127 (2013.01)
(72) 발명자
임채민
서울특별시 서대문구 연세로 50
이수현
서울특별시 서대문구 연세로 50
송도경
서울특별시 서대문구 연세로 50
이진호
서울특별시 마포구 월드컵북로 235

이 발명을 지원한 국가연구개발사업

과제고유번호	1711193986
과제번호	2020-0-01361-004
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원
연구사업명	정보통신방송혁신인재양성
연구과제명	인공지능대학원지원(연세대학교)
기 여 율	1/2
과제수행기관명	연세대학교 산학협력단
연구기간	2023.01.01 ~ 2023.12.31

이 발명을 지원한 국가연구개발사업

과제고유번호	1415177659
과제번호	P0016150
부처명	산업통상자원부
과제관리(전문)기관명	한국산업기술진흥원
연구사업명	산업기술국제협력(R&D)
연구과제명	스마트 IoT 네트워크 보안을 위한 공격탐지 및 방어 기술 개발
기 여 율	1/2
과제수행기관명	한국전자기술연구원
연구기간	2021.12.01 ~ 2022.11.30

명 세 서

청구범위

청구항 1

암호화 입력 데이터를 송신하고 암호화 출력 데이터를 수신하는 제1 유형의 메모리; 및

트러스트 신뢰 공간에서 동작되고 상기 암호화 입력 및 출력 데이터의 인공지능 연산을 수행하는 트러스트 인공지능 처리부를 포함하고,

상기 트러스트 인공지능 처리부는

상기 암호화 입력 데이터의 복호화를 통해 복호화 입력 데이터를 생성하고 상기 암호화 출력 데이터를 생성하기 위해 비암호화 출력 데이터의 암호화를 수행하는 암호처리 프론트-엔드 프로세서;

상기 복호화 입력 데이터 및 상기 비암호화 출력 데이터에 대한 버퍼를 제공하는 제2 유형의 메모리; 및

상기 복호화 입력 데이터를 기초로 신경망 연산을 수행하여 상기 비암호화 출력 데이터를 생성하는 프로세서를 포함하는 트러스트 환경 기반 인공지능 장치.

청구항 2

제1항에 있어서, 상기 암호처리 프론트-엔드 프로세서는

상기 제1 유형의 메모리로부터 상기 암호화 입력 데이터로서 암호화 입력 액티베이션 및 암호화 필터를 입력받아 상기 제2 유형의 메모리에 복호화 입력 액티베이션 및 복호화 필터를 저장하는 것을 특징으로 하는 트러스트 환경 기반 인공지능 장치.

청구항 3

제1항에 있어서, 상기 암호처리 프론트-엔드 프로세서는

상기 인공지능 연산의 과정에서 상기 프로세서에 의한 온-디맨드 요청을 수신하고 상기 제1 유형의 메모리에 접근하여 상기 암호화 입력 데이터를 가져오는 것을 특징으로 하는 트러스트 환경 기반 인공지능 장치.

청구항 4

제1항에 있어서, 상기 제2 유형의 메모리는

상기 제1 유형의 메모리 보다 상대적으로 빠른 동작속도 및 적은 저장용량을 가지는 것을 특징으로 하는 트러스트 환경 기반 인공지능 장치.

청구항 5

제1항에 있어서, 상기 프로세서는

직접 컨볼루션 기반의 신경망 연산을 수행하여 상기 제1 유형의 메모리의 접근 횟수를 줄이는 것을 특징으로 하는 트러스트 환경 기반 인공지능 장치.

청구항 6

제5항에 있어서, 상기 프로세서는

상기 제2 유형의 메모리에 복호화 입력 액티베이션을 고정적으로 저장하고 복호화 필터 및 비암호 출력 액티베이션을 순환큐 방식으로 저장하는 것을 특징으로 하는 트러스트 환경 기반 인공지능 장치.

청구항 7

제1항에 있어서, 상기 프로세서는

상기 암호처리 프론트-엔드 프로세서의 인터럽트 지향 오프로딩(interrupt driven offloading)을 통해 상기 제1 및 제2 유형의 메모리들과 데이터 송수신을 수행하는 것을 특징으로 하는 트러스트 환경 기반 인공지능 장치.

청구항 8

제1항에 있어서, 상기 프로세서는

DMA (Direct Memory Access) 컨트롤러의 DMA 지향 오프로딩을 통해 상기 암호처리 프론트-엔드 프로세서와 상기 제1 및 제2 유형의 메모리들과 데이터 송수신을 수행하는 것을 특징으로 하는 트러스트 환경 기반 인공지능 장치.

청구항 9

제1항에 있어서, 상기 프로세서는

상기 암호처리 프론트-엔드 프로세서의 암호화 및 복호화 연산들의 수행과 오버랩핑되게 상기 신경망 연산의 수행을 수행하여 인트라-레이어 파이프라이닝을 구현하는 것을 특징으로 하는 트러스트 환경 기반 인공지능 장치.

청구항 10

제9항에 있어서, 상기 프로세서는

상기 신경망 연산의 수행 중간에 상기 암호처리 프론트-엔드 프로세서가 상기 암호화 입력 데이터의 복호화 연산을 수행하도록 하여 상기 신경망 연산을 끊김없이 수행하는 것을 특징으로 하는 트러스트 환경 기반 인공지능 장치.

청구항 11

제9항에 있어서, 상기 프로세서는

상기 인트라-레이어 파이프라이닝을 위해 데이터 복호화 단계, 연산 단계 및 데이터 암호화 단계로 세분화하여 상기 신경망 연산을 끊김없이 수행하는 것을 특징으로 하는 트러스트 환경 기반 인공지능 장치.

청구항 12

암호화 입력 데이터를 송신하는 제1 유형의 메모리; 및

트러스트 신뢰 공간에서 동작되고 상기 암호화 입력 데이터의 인공지능 연산을 수행하는 트러스트 인공지능 처리부를 포함하고,

상기 트러스트 인공지능 처리부는

상기 암호화 입력 데이터의 복호화를 통해 복호화 입력 데이터를 생성하는 암호처리 프론트-엔드 프로세서;

상기 복호화 입력 데이터 및 상기 비암호화 출력 데이터에 대한 버퍼를 제공하는 제2 유형의 메모리; 및

상기 복호화 입력 데이터를 기초로 신경망 연산을 수행하여 상기 비암호화 출력 데이터를 생성하는 프로세서를 포함하는 트러스트 환경 기반 인공지능 장치.

청구항 13

제12항에 있어서, 상기 프로세서는

직접 컨볼루션 기반의 신경망 연산을 수행하여 상기 제1 유형의 메모리의 접근 횟수를 줄이는 것을 특징으로 하는 트러스트 환경 기반 인공지능 장치.

청구항 14

제12항에 있어서, 상기 프로세서는

상기 제2 유형의 메모리에 복호화 입력 액티베이션을 고정적으로 저장하고 복호화 필터 및 비암호 출력 액티베이션을 순환큐 방식으로 저장하는 것을 특징으로 하는 트러스트 환경 기반 인공지능 장치.

청구항 15

제12항에 있어서, 상기 프로세서는

상기 암호처리 프론트-엔드 프로세서의 암호화 및 복호화 연산들의 수행과 오버랩핑되게 상기 신경망 연산의 수행을 수행하여 인트라-레이어 파이프라이닝을 구현하는 것을 특징으로 하는 트러스트 환경 기반 인공지능 장치.

발명의 설명

기 술 분 야

[0001] 본 발명은 인공지능 신경망 프레임워크 기술에 관한 것으로, 보다 상세하게는 트러스트(Trust) 환경에서 인공신경망 실행을 안전하게 가속화할 수 있는 트러스트 환경 기반 인공지능 장치에 관한 것이다.

배 경 기 술

[0003] DNN(Deep Neural Network)은 입력층과 출력층 사이에 여러 개의 은닉층들로 이뤄진 인공신경망이다. DNN은 모바일 및 임베디드 애플리케이션에서 널리 사용되고 있다. 특히, DNN은 사용자의 신원 검증을 위해 사용자의 생물학적 특성(예: 지문, 홍채, 얼굴 등)을 사용하는 생체 인증 수행 애플리케이션에 유용하다.

[0004] DNN에는 민감한 사용자 데이터가 많이 포함되어 있으므로 모바일 및 임베디드 기기는 보안 공격으로부터 사용자와 DNN 데이터를 안전하게 보호할 수 있는 안전한 DNN 실행 환경을 구현해야 한다.

[0005] 기존에는 ARM(Advanced RISC Machine) 프로세서에서 사용할 수 있는 하드웨어 기반 보안 기술인 트러스트 존(TrustZone)을 통해 트러스트 실행 환경(Trusted Execution Environment)에서 DNN을 실행할 것이 제안되었다. 트러스트 존은 프로세서(processor) 안에 독립적인 보안 구역을 따로 두어 중요한 정보를 보호하는 하드웨어 기반의 보안 기술이다. 하지만, 트러스트 존에서 DNN을 실행하는 것만으로는 데이터를 완전히 보호할 수 없다. 트러스트 존은 메모리 보호 기능이 제한되어 있기 때문이다. 트러스트 존을 사용하면 하드웨어 및 소프트웨어 리소스를 안전한 일반 구역으로 분할하여 DNN 실행을 다른 프로세스와 분리할 수 있다. 일반 구역에서 보안 구역의 데이터에 액세스하는 것은 하드웨어에 의해 방지될 수 있다. 그러나 트러스트 존은 데이터를 휘발성 메모리에 암호화하지 않은 상태로 유지하기 때문에 콜드 부트 공격(Cold Boot Attack)과 같은 물리적 보안 공격은 트러스트 존에서 DNN 실행에도 불구하고 민감한 사용자 및 DNN 데이터를 획득할 수 있다.

[0006] 물리적 공격으로부터 데이터를 보호하기 위해 데이터를 암호화하고 안전한 온칩 메모리에서만 복호화하도록 선택할 수 있다. 이러한 방식은 트러스트 존을 사용하여 DNN 실행을 다른 프로세스로부터 격리할 수 있을 뿐만 아니라 암호화를 통해 물리적 공격으로부터 사용자 및 DNN 데이터를 보호할 수 있다. 하지만, 메모리에서 암호화된 데이터를 보호하면 느린 메모리 액세스가 크게 증가하고 프로세서에 부과되는 높은 데이터 암복호화 오버헤드로 인해 DNN 실행 시간이 크게 증가하는 문제가 발생할 수 있다.

[0007] 따라서 민감한 사용자와 DNN 데이터를 물리적 공격으로부터 보호할 뿐만 아니라 느린 메모리 액세스와 높은 데이터 암복호화 오버헤드를 극복하여 DNN 실행 시간을 줄이는 새로운 보안 DNN 프레임워크가 필요하게 되었다.

선행기술문헌

특허문헌

[0009] (특허문헌 0001) 한국등록특허 제10-2474875호 (2022.12.01)

발명의 내용

해결하려는 과제

[0010] 본 발명의 일 실시예는 트러스트(Trust) 환경에서 인공신경망 실행을 안전하게 가속화할 수 있는 트러스트 환경 기반 인공지능 장치를 제공하고자 한다.

[0011] 본 발명의 일 실시예는 트러스트 신뢰 공간에서 데이터의 암호화를 수행하여 물리적 공격으로부터 보안을 강화하고 직접 컨볼루션 기반의 신경망 연산을 수행하여 메모리 접근 횟수를 줄일 수 있는 트러스트 환경 기반 인공지능 장치를 제공하고자 한다.

[0012] 본 발명의 일 실시예는 암호 하드웨어로 오프로딩을 수행하여 신경망 실행 작업이 제한된 프로세서 리소스를 활용할 수 있으며 인트라-레이어 파이프라이닝을 통해 데이터 암호화 및 복호화와 오버랩핑하여 인공신경망 실행 시간을 단축할 수 있는 트러스트 환경 기반 인공지능 장치를 제공하고자 한다.

과제의 해결 수단

[0014] 실시예들 중에서, 트러스트 환경 기반 인공지능 장치는 암호화 입력 데이터를 송신하고 암호화 출력 데이터를 수신하는 제1 유형의 메모리; 및 트러스트 신뢰 공간에서 동작되고 상기 암호화 입력 및 출력 데이터의 인공지능 연산을 수행하는 트러스트 인공지능 처리부를 포함하고, 상기 트러스트 인공지능 처리부는 상기 암호화 입력 데이터의 복호화를 통해 복호화 입력 데이터를 생성하고 상기 암호화 출력 데이터를 생성하기 위해 비암호화 출력 데이터의 암호화를 수행하는 암호처리 프론트-엔드 프로세서; 상기 복호화 입력 데이터 및 상기 비암호화 출력 데이터에 대한 버퍼를 제공하는 제2 유형의 메모리; 및 상기 복호화 입력 데이터를 기초로 신경망 연산을 수행하여 상기 비암호화 출력 데이터를 생성하는 프로세서를 포함한다.

[0015] 상기 암호처리 프론트-엔드 프로세서는 상기 제1 유형의 메모리로부터 상기 암호화 입력 데이터로서 암호화 입력 액티베이션 및 암호화 필터를 입력받아 상기 제2 유형의 메모리에 복호화 입력 액티베이션 및 복호화 필터를 저장할 수 있다.

[0016] 상기 암호처리 프론트-엔드 프로세서는 상기 인공지능 연산의 과정에서 상기 프로세서에 의한 온-디맨드 요청을 수신하고 상기 제1 유형의 메모리에 접근하여 상기 암호화 입력 데이터를 가져올 수 있다.

[0017] 상기 제2 유형의 메모리는 상기 제1 유형의 메모리 보다 상대적으로 빠른 동작속도 및 적은 저장용량을 가질 수 있다.

[0018] 상기 프로세서는 직접 컨볼루션 기반의 신경망 연산을 수행하여 상기 제1 유형의 메모리의 접근 횟수를 줄일 수 있다.

[0019] 상기 프로세서는 상기 제2 유형의 메모리에 복호화 입력 액티베이션을 고정적으로 저장하고 복호화 필터 및 비

암호 출력 액티베이션을 순환큐 방식으로 저장할 수 있다.

[0020] 상기 프로세서는 상기 암호처리 프론트-엔드 프로세서의 인터럽트 지향 오프로딩(interrupt driven offloading)을 통해 상기 제1 및 제2 유형의 메모리들과 데이터 송수신을 수행할 수 있다.

[0021] 상기 프로세서는 DMA (Direct Memory Access) 컨트롤러의 DMA 지향 오프로딩을 통해 상기 암호처리 프론트-엔드 프로세서와 상기 제1 및 제2 유형의 메모리들과 데이터 송수신을 수행할 수 있다.

[0022] 상기 프로세서는 상기 암호처리 프론트-엔드 프로세서의 암호화 및 복호화 연산들의 수행과 오버랩핑되게 상기 신경망 연산의 수행을 수행하여 인트라-레이어 파이프라이닝을 구현할 수 있다.

[0023] 상기 프로세서는 상기 신경망 연산의 수행 중간에 상기 암호처리 프론트-엔드 프로세서가 상기 암호화 입력 데이터의 복호화 연산을 수행하도록 하여 상기 신경망 연산을 끊김없이 수행할 수 있다.

[0024] 상기 프로세서는 상기 인트라-레이어 파이프라이닝을 위해 데이터 복호화 단계, 연산 단계 및 데이터 암호화 단계로 세분화하여 상기 신경망 연산을 끊김없이 수행할 수 있다.

[0025] 실시예들 중에서, 트러스트 환경 기반 인공지능 장치는 암호화 입력 데이터를 송신하는 제1 유형의 메모리; 및 트러스트 신뢰 공간에서 동작되고 상기 암호화 입력 데이터의 인공지능 연산을 수행하는 트러스트 인공지능 처리부를 포함하고, 상기 트러스트 인공지능 처리부는 상기 암호화 입력 데이터의 복호화를 통해 복호화 입력 데이터를 생성하는 암호처리 프론트-엔드 프로세서; 상기 복호화 입력 데이터 및 상기 비암호화 출력 데이터에 대한 버퍼를 제공하는 제2 유형의 메모리; 및 상기 복호화 입력 데이터를 기초로 신경망 연산을 수행하여 상기 비암호화 출력 데이터를 생성하는 프로세서를 포함한다.

[0026] 상기 프로세서는 직접 컨볼루션 기반의 신경망 연산을 수행하여 상기 제1 유형의 메모리의 접근 횟수를 줄일 수 있다.

[0027] 상기 프로세서는 상기 제2 유형의 메모리에 복호화 입력 액티베이션을 고정적으로 저장하고 복호화 필터 및 비암호 출력 액티베이션을 순환큐 방식으로 저장할 수 있다.

[0028] 상기 프로세서는 상기 암호처리 프론트-엔드 프로세서의 암호화 및 복호화 연산들의 수행과 오버랩핑되게 상기 신경망 연산의 수행을 수행하여 인트라-레이어 파이프라이닝을 구현할 수 있다.

발명의 효과

[0030] 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.

[0031] 본 발명에 따른 트러스트 환경 기반 인공지능 장치는 트러스트(Trust) 환경에서 인공신경망 실행을 안전하게 가속화할 수 있다.

[0032] 본 발명에 따른 트러스트 환경 기반 인공지능 장치는 트러스트 신뢰 공간에서 데이터의 암호화를 수행하여 물리적 공격으로부터 보안을 강화하고 직접 컨볼루션 기반의 신경망 연산을 수행하여 메모리 접근 횟수를 줄일 수 있다.

[0033] 본 발명에 따른 트러스트 환경 기반 인공지능 장치는 암호 하드웨어로 오프로딩을 수행하여 신경망 실행 작업이 제한된 프로세서 리소스를 활용할 수 있으며 인트라-레이어 파이프라이닝을 통해 데이터 암호화 및 복호화와 오버랩핑하여 인공신경망 실행 시간을 단축할 수 있다.

도면의 간단한 설명

[0035] 도 1은 본 발명에 따른 트러스트 환경 기반 인공지능 장치를 설명하는 도면이다.

도 2는 도 1의 인공지능 장치에서 트러스트 환경 기반 인공지능 연산의 수행 방법을 설명하는 순서도이다.

도 3은 트러스트 실행 환경을 설명하는 도면이다.

도 4a-4b는 기존 대비 본 발명에 따른 DNN 프레임워크의 작업 모델을 설명하는 도면이다.

도 5 내지 도 7은 기존 DNN 프레임워크의 실행 속도, 대역폭 및 복호화 처리량을 각각 보여주는 도면이다.

도 8a-8b는 신경망 연산 수행에 사용되는 컨볼루션을 설명하는 도면이다.

도 9a-9b는 DNN 친화적인 SRAM 관리 과정을 설명하는 도면이다.

도 10은 CPU와 암호 하드웨어 간의 처리량을 보여주는 도면이다.

도 11은 암호 하드웨어의 오프로딩을 설명하는 도면이다.

도 12 및 13은 신경망 연산의 수행 과정에서 인트라-레이어 파이프라이닝의 구현을 설명하는 도면이다.

도 14는 본 발명의 소프트웨어 구성 예를 설명하는 도면이다.

도 15 내지 17은 본 발명에 따른 실험 결과를 설명하는 도면이다.

발명을 실시하기 위한 구체적인 내용

[0036] 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.

[0037] 한편, 본 출원에서 서술되는 용어의 의미는 다음과 같이 이해되어야 할 것이다.

[0038] "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.

[0039] 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.

[0040] 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.

[0041] 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.

[0042] 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.

[0043] 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.

[0045] 도 1은 본 발명에 따른 트러스트 환경 기반 인공지능 장치를 설명하는 도면이다.

[0046] 도 1을 참조하면, 트러스트 환경 기반 인공지능 장치(100)는 모바일 또는 임베디드 기기에서 인공신경망 실행에 대해 보안 강화 및 가속화를 위한 프레임워크로서, 제1 유형의 메모리(110)와 트러스트 인공지능 처리부(130)를 포함하여 구현될 수 있다.

[0047] 제1 유형의 메모리(110)는 암호화 입력 데이터를 송신하고 암호화 출력 데이터를 수신할 수 있다. 여기에서, 제1 유형의 메모리(110)는 DRAM(Dynamic Random Access Memory)으로 구성될 수 있으며, 반드시 이에 한정되는 것은 아니다.

[0048] 트러스트 인공지능 처리부(130)은 트러스트 신뢰 공간에서 동작되고 암호화 입력 및 출력 데이터의 인공지능 연산을 수행할 수 있다. 이를 위해, 트러스트 인공지능 처리부(130)는 암호처리 프론트-엔드 프로세서(131), 제2 유형의 메모리(133) 및 프로세서(135)를 포함할 수 있다.

[0049] 암호처리 프론트-엔드 프로세서(131)는 암호 하드웨어(Crytographic Hardware)로서, 암호화 입력 데이터의 복호화를 통해 복호화 입력 데이터를 생성하고 암호화 출력 데이터를 생성하기 위해 비암호화 출력 데이터의 암호화를 수행할 수 있다. 암호처리 프론트-엔드 프로세서(131)는 제1 유형의 메모리(110)로부터 암호화 입력 데이터로서 암호화 입력 액티베이션 및 암호화 필터를 입력받아 제2 유형의 메모리(133)에 복호화 입력 액티베이션 및 복호화 필터를 저장할 수 있다. 암호처리 프론트-엔드 프로세서(131)는 인공지능 연산의 과정에서 프로세서(135)에 의한 온-디맨드(On-Demand) 요청을 수신하고 제1 유형의 메모리(110)에 접근하여 암호화 입력 데이터를 가져올 수 있다.

[0050] 제2 유형의 메모리(133)는 복호화 입력 데이터 및 비암호화 출력 데이터에 대한 버퍼를 제공할 수 있다. 여기에서, 제2 유형의 메모리(133)는 제1 유형의 메모리(110) 보다 상대적으로 빠른 동작속도 및 적은 저장용량을 가질 수 있다. 예를 들어, 제1 유형의 메모리(110)가 DRAM으로 구성된 경우, 제2 유형의 메모리(133)는 SRAM(Static Random Access Memory)으로 구성될 수 있다. 제1 및 제2 유형의 메모리(110,133) 사이에 데이터를 송수신할 때 암호처리 프론트-엔드 프로세서(131)에서 데이터를 암호화 및 복호화하여 정확성 및 보안성을 보장할 수 있다.

[0051] 프로세서(135)는 복호화 입력 데이터를 기초로 신경망 연산을 수행하여 비암호화 출력 데이터를 생성할 수 있다. 프로세서(135)는 직접 컨볼루션 기반의 신경망 연산을 수행하여 제1 유형의 메모리(110)의 접근 횟수를 줄일 수 있다. 즉, 프로세서(135)는 직접 컨볼루션을 사용하여 신경망 연산의 작업 세트 크기를 최소화하고 감소된 작업 세트 크기로 인해 신경망 연산을 수행 중 제2 유형의 메모리(133)의 동작속도 보다 상대적으로 느린 제1 유형의 메모리(110)에 대한 접근 및 데이터 암호화 및 복호화 호출이 줄어들 수 있다.

[0052] 프로세서(135)는 제2 유형의 메모리(133)에 복호화 입력 액티베이션을 고정적으로 저장하고 복호화 필터 및 비암호 출력 액티베이션을 순환큐 방식으로 저장할 수 있다. 프로세서(135)는 신경망 연산의 수행에서 컨볼루션 레이어의 입력 액티베이션, 필터 및 출력 액티베이션의 접근 패턴을 기초로 제2 유형의 메모리(133)를 효율적으로 관리할 수 있다. 프로세서(135)는 입력 액티베이션의 경우 데이터의 재사용을 증가시키는 경향이 있기 때문에 고정적으로 저장하고, 필터의 경우 출력 채널을 생성할 때 동일한 필터를 여러 번 사용하지만 크기가 입력 액티베이션보다 훨씬 작고 하나의 출력 채널만 생성하는 데 필요하기 때문에 순환큐(Circular Queue) 방식으로 저장하고, 출력 액티베이션의 경우 공간적 집약성이 높은 1회 쓰기 데이터이므로 순환큐 방식으로 저장할 수 있다.

[0053] 프로세서(135)는 암호처리 프론트-엔드 프로세서(131)의 인터럽트 지향 오프로딩(interrupt driven offloading)을 통해 제1 및 제2 유형의 메모리들(110,131)과 데이터 송수신을 수행할 수 있다. 또한, 프로세서(135)는 DMA(Direct Memory Access) 컨트롤러의 DMA 지향 오프로딩을 통해 암호처리 프론트-엔드 프로세서(131)와 제1 및 제2 유형의 메모리들(110,133)과 데이터 송수신을 수행할 수 있다. 프로세서(135)는 인터럽트 지향 오프로딩 및 DMA 지향 오프로딩 중에서 선택할 수 있다. 인터럽트 지향 오프로딩은 대기시간이 중요한 작업을 수행하는 데 적합하고, DMA 지향 오프로딩은 한번에 더 많은 양의 데이터를 처리하는 데 적합한다. 여기에서, 두 오프로딩 메커니즘은 신경망 연산을 가속화할 수 있다.

[0054] 프로세서(135)는 암호처리 프론트-엔드 프로세서(131)의 암호화 및 복호화 연산들의 수행과 오버랩핑되게 신경망 연산의 수행을 수행하여 인트라-레이어 파이프라이닝을 구현할 수 있다. 프로세서(135)는 신경망 연산의 수

행 중간에 암호처리 프론트-엔드 프로세서(131)가 암호화 입력 데이터의 복호화 연산을 수행하도록 하여 신경망 연산을 끊김없이 수행할 수 있다. 프로세서(135)는 인트라-레이어 파이프라이닝을 위해 데이터 복호화 단계, 연산 단계 및 데이터 암호화 단계로 세분화하여 신경망 연산을 끊김없이 수행할 수 있다. 프로세서(135)는 인트라-레이어 파이프라이닝을 통해 데이터 복호화 단계, 연산 단계 및 데이터 암호화 단계를 병렬화할 수 있다.

[0056] 도 2는 도 1의 인공지능 장치에서 트러스트 환경 기반 인공지능 연산의 수행 방법을 설명하는 순서도이다.

[0057] 도 2에서, 인공지능 장치(100)는 제1 유형의 메모리(110)로부터 암호화 입력 데이터를 입력받아 복호화를 통해 복호화 입력 데이터를 생성한다(단계 S210). 인공지능 장치(100)는 암호처리 프론트-엔드 프로세서(131)를 통해 암호화 입력 데이터에 대해 복호화를 수행할 수 있다. 인공지능 장치(100)는 복호화 입력 데이터를 제2 유형의 메모리(133)에 저장할 수 있다.

[0058] 인공지능 장치(100)는 복호화 입력 데이터를 기초로 신경망 연산을 수행하여 비암호화 출력 데이터를 생성한다(단계 S230). 여기에서, 인공지능 장치(100)는 프로세서(135)를 통해 직접 컨볼루션 기반의 신경망 연산을 수행하여 암호처리 프론트-엔드 프로세서(131)가 제1 유형의 메모리(110)에 접근하여 암호화 입력 데이터를 가져오는 접근 횟수를 줄일 수 있다. 인공지능 장치(100)는 비암호화 출력 데이터를 제2 유형의 메모리(133)에 저장할 수 있다.

[0059] 인공지능 장치(100)는 비암호화 출력 데이터의 암호화를 수행하여 암호화 출력 데이터를 생성한다(단계 S250). 인공지능 장치(100)는 암호처리 프론트-엔드 프로세서(310)를 통해 비암호화 출력 데이터에 대해 암호화를 수행할 수 있다. 인공지능 장치(100)는 암호화 출력 데이터를 제1 유형의 메모리(110)로 출력할 수 있다.

[0060] 인공지능 장치(100)는 암호처리 프론트-엔드 프로세서(131)의 암호화 및 복호화 연산들의 수행과 오버랩핑되게 프로세서(135)의 신경망 연산의 수행을 수행하여 데이터 복호화 단계, 연산 단계 및 데이터 암호화 단계로 인트라-레이어 파이프라이닝을 구현함으로써 신경망 연산을 가속화할 수 있다.

[0062] 이하, 도 3 내지 17을 참조하여 본 발명에 따른 트러스트 환경 기반 인공지능 장치에 대해 보다 자세히 설명한다.

[0063] 도 3은 트러스트 실행 환경을 설명하는 도면이다.

[0064] 도 3을 참조하면, 트러스트 실행 환경(Trusted Execution Environment; 이하, TEE 라 함)은 처리, 메모리 및 저장 기능을 갖춘 보안 처리 환경으로, 민감한 작업과 데이터가 해당 환경을 벗어나지 않도록 제한하여 높은 보안을 달성할 수 있다. TEE 내의 작업 및 데이터는 OS(Operating System) 및 애플리케이션이 실행되는 리치 실행 환경(Rich Execution Environment; 이하, REE 라 함)에서 격리될 수 있다.

[0065] 모바일 및 임베디드 장치에서 사용할 수 있는 트러스트존 지원 TEE를 구현하기 위해 트러스트존은 CPU가 지정된 시점에 TEE 및 REE 중 하나에서만 독점적으로 작동하도록 하는 보안 프로세서 모드를 구현한다. TEE 및 REE 간의 전환 및 상호 작용은 SMC(Secure Monitor Calls)로 호출할 수 있는 보안 모니터에 의해 관리된다. 또한, 트러스트존은 DRAM을 보안 및 비보안 영역으로 분할하고, TEE의 민감한 데이터를 보호하기 위해 REE에서 보안 영역에 접근하는 것을 허용하지 않는다.

[0066] 트러스트존 지원 TEE에서 실행하는 기존의 DNN 프레임워크는, ① 주변장치로부터 입력 데이터(예: 지문 센서의 지문 이미지)를 수신하여 REE에서 시작한다. ② REE에서 DNN의 몇 가지 초기 레이어를 실행한 후, ③ 출력 액티베이션을 암호화하고 SMC를 통해 TEE로 전송한다. 그런 다음, ④ TEE 내에서 전송된 액티베이션을 복호화하고, ⑤ 복호화된 액티베이션과 사전 전송된 필터를 사용하여 나머지 레이어를 실행한다. 나머지 레이어의 실행을 완료하면, ⑥ DNN이 만든 예측을 SMC를 통해 REE로 반환한다.

[0067] 이러한 방식으로 DNN 실행을 격리하고 몇 가지 보안 공격으로부터 DNN을 보호할 수 있다.

[0068] 도 4a 및 4b는 기존 대비 본 발명에 따른 DNN 프레임워크의 작업 모델을 설명하는 도면으로, 도 4a는 기존 DNN 프레임워크이고, 도 4b는 본 발명에서 제안한 DNN 프레임워크(이하, GuardianNN 이라 함)이다.

[0069] 도 4a의 기존 DNN 프레임워크의 경우, 처음에는 DNN의 입력 데이터와 필터가 모두 암호화되어 DRAM에 저장된다. 프레임워크가 DNN의 레이어 실행을 시작하면 ① 암호화된 입력 액티베이션 및 필터를 SRAM에 로드하고 ② CPU를

사용하여 데이터를 복호화한다. 그런 다음, 프레임워크는 ③ 복호화된 데이터를 사용하여 각 레이어를 실행하고 출력 액티베이션을 SRAM에 저장한다. 그후 프레임워크는 ④ 출력 액티베이션을 암호화하고 ⑤ 이를 DRAM에 저장한다. 입력 액티베이션 및 필터는 필요에 따라 SRAM으로 교체된다. SRAM이 레이어의 작업 집합보다 작으면 데이터 스왑핑 및 암복호화가 자주 발생한다. 암호화된 DRAM을 사용하고 TEE 내에서 DNN을 실행하면 기존 DNN 프레임워크가 보안 공격으로부터 민감한 사용자 및 DNN 데이터를 완전히 보호할 수 있지만, 느린 DNN 실행으로 어려움을 겪을 수 있다.

[0070] 도 5 내지 도 7은 기존 DNN 프레임워크의 실행 속도, 대역폭 및 복호화 처리량을 각각 보여주는 도면이다.

[0071] 도 5에 보여진 바와 같이, 도 4a의 기존 DNN 프레임워크(OP-TEE with Pager)와 DRAM 데이터를 암호화하지 않는 안전하지 않은 DarkneTZ 작업모델의 DNN 실행 속도를 비교한 결과로서, 기존 DNN 프레임워크의 DNN 실행 속도가 DarkneTZ보다 현저히 느리게 나타났다. 기존 DNN 프레임워크는 DarkneTZ 보다 AlexNet을 실행하는 데 3.42배 더 오래 걸렸다.

[0072] 기존 DNN 프레임워크는 임베디드 SRAM의 제한된 용량으로 인해 느린 DRAM 접근 횟수가 증가하고, CPU에 부과되는 높은 데이터 암호화 및 복호화 오버헤드로 인한 성능 병목 현상이 DNN 실행 속도에 영향을 미칠 수 있다. 구체적으로, 기존 DNN 프레임워크는 TEE가 암호화된 DRAM 데이터를 안전하게 로드하고 로드된 데이터를 복호화할 수 있는 안전한 온칩 메모리로 임베디드 SRAM을 활용한다. 그러나 SRAM의 용량(수백 KB)은 일반적으로 모바일 및 임베디드 장치의 온칩 CPU 캐시(수백 KB)보다 작기 때문에 SRAM을 안전한 온칩 버퍼로 사용한다. 효과적인 온칩 메모리 크기를 한 자릿수 줄인다. 이로 인해 DNN을 실행할 때 느린 DRAM 접근 횟수가 증가하여 DNN 실행 속도가 느려진다.

[0073] 도 6에 보여진 바와 같이, 작업 세트 크기가 다양한 보안 내장형 SRAM과 오프칩 DRAM의 대역폭을 비교한 결과, SRAM이 다양한 작업 세트 크기에 대해 지속적으로 DRAM보다 더 높은 대역폭을 제공하는 것으로 나타났다. 이는 증가된 DRAM 액세스가 DNN 실행 속도에 부정적인 영향을 미치는 것을 의미한다. 결과는 또한 더 높은 대역폭을 달성하기 위해 순차 액세스가 임의 액세스보다 선호되어야 함을 시사한다. 결과적으로, 느린 DRAM 액세스가 DNN 실행 속도에 미치는 부정적인 영향을 최소화하려면 빠른 SRAM 데이터의 재사용을 최대화하고 DNN 실행의 메모리 액세스를 순차 액세스로 구성해야 한다.

[0074] 임베디드 SRAM과 암호화된 DRAM 간에 데이터를 교환할 때 기능적 정확성과 높은 보안성을 보장하기 위해 CPU에서 데이터를 암호화 및 복호화해야 한다. 하지만, CPU 기반 암복호화는 속도가 느릴 뿐만 아니라 제한된 CPU 대역폭의 상당 부분을 소모하기 때문에 제한된 CPU 대역폭이 컴퓨팅 집약적인 DNN 실행과 암호화 및 복호화 모두에서 공유되어 DNN 실행 속도가 느려진다.

[0075] 도 7에 보여진 바와 같이, 페이로드 크기가 다양한 CPU 기반 데이터 복호화 처리량은 데이터 암호화 및 복호화가 제한된 CPU 리소스로 인해 어려움을 겪고 있으며 2KB 페이로드로 4.51MB/s의 처리량을 달성한다는 것을 나타낸다. 2KB의 DRAM 데이터에 순차적으로 액세스하는 처리량(609.24MB/s, 도 6의 (a))과 비교할 때 CPU 기반 2KB 데이터 암호화 및 복호화는 순차적 DRAM 액세스보다 135.09배 느리다. 낮은 데이터 암호화 및 복호화 처리량은 데이터 암호화 및 복호화의 오버헤드가 상당하여 기존 DNN 프레임워크가 빠른 DNN 실행을 달성하기 어렵다는 것을 나타낸다. 결과적으로, 기존 DNN 프레임워크는 제한된 CPU 리소스에 부과되는 높은 DRAM 데이터 암호화 및 복호화 오버헤드로 인해 상당한 어려움을 겪고 있으며, 빠르고 안전한 DNN 실행을 위해서는 오버헤드를 해결해야 한다.

[0076] 이에, 본 발명은 기존 DNN 프레임워크의 느린 DNN 실행 문제를 해결하여 빠르고 안전한 모바일 및 임베디드 장치용 DNN 프레임워크인 GuardiaNN을 제안한다. 본 발명에서 제안한 GuardiaNN은 다음과 같은 특징을 갖는다.

[0077] - 직접 컨볼루션을 통해 DNN 실행의 작업 세트 크기를 최소화할 수 있다. 작업 세트 크기를 감소시켜 DNN 실행 중 느린 DRAM 액세스 및 데이터 암호화 및 복호화 요청이 줄어들 수 있다.

[0078] - DNN 친환적인 SRAM 관리를 사용하여 컨볼루션 레이어의 데이터 재사용을 최대화를 할 수 있다. 입력 액티베이션을 SRAM에 고정하고 필터 및 출력 액티베이션을 순환큐 방식으로 저장하여 SRAM을 효율적으로 관리할 수 있다.

[0079] - 암호 하드웨어로 데이터 암호화 및 복호화를 오프로드할 수 있다. 데이터 암호화 및 복호화를 오프로드하면 제한된 CPU 리소스를 DNN 전용으로 사용할 수 있다.

[0080] - DNN 레이어와 데이터 암호화 및 복호화 작업을 오버랩핑하여 DNN 실행을 더욱 가속화할 수 있다. 이는 CPU와

암호 하드웨어가 각각 DNN과 데이터 암호화 및 복호화 작업을 동시에 수행할 수 있기 때문에 가능하다.

[0081] 도 4b를 보면, 본 발명에서 제안한 DNN 프레임워크의 작업모델은 DNN 레이어를 실행할 때 ① 모바일 및 임베디드 장치에서 사용할 수 있는 암호 하드웨어(도 1의 암호처리 프론트-엔드 프로세서)에서 DRAM에서 암호화된 입력 액티베이션 및 필터를 로드하고 데이터를 복호화하고 복호화 데이터를 SRAM에 저장한다. 그런 다음, ② CPU(도 1의 프로세서)는 SRAM 데이터를 사용하여 DNN 레이어의 작업을 수행하고 출력 액티베이션을 SRAM(도 1의 제2 유형의 메모리)에 저장한다. 그후 ③ 암호 하드웨어는 SRAM에 저장된 출력 액티베이션을 암호화하고 암호화된 출력 액티베이션을 DRAM(도 1의 제1 유형의 메모리)에 저장한다.

[0082] 도 4a의 기존 DNN 프레임워크의 작업 모델과 비교할 때, 직접 컨볼루션 및 DNN 친화적인 SRAM 관리를 사용하여 느린 DRAM 액세스와 암호 하드웨어를 사용하여 CPU에 부과되는 높은 데이터 암호화 및 복호화 오버헤드를 크게 줄일 수 있다. 본 발명에서 제안한 주요 특징들이 DNN 실행에 미치는 영향은 하기 표 1로 정리할 수 있다.

[0083] [표 1]

Key Idea	Beneficial to DNNs Having
Direct Convolutions	Large filter sizes, high input channel counts, small stride sizes
DNN-Friendly SRAM Mgmt.	High output channel counts
Cryptographic Hardware	Large memory footprint sizes
Intra-Layer Pipelining	Balanced compute & encryption/decryption

[0084]

[0085] 본 발명은 직접 컨볼루션을 사용하여 컨볼루션 레이어의 작업 세트 크기를 줄이고, DNN 친화적인 SRAM 관리를 사용하여 SRAM 데이터 재사용을 최대화하여 DNN 실행 중 느린 DRAM 액세스를 크게 줄일 수 있다.

[0086] 도 8a 및 8b는 신경망 연산 수행에 사용되는 컨볼루션을 설명하는 도면으로, 도 8a는 im2col(Image to Column) 컨볼루션이고, 도 8b는 직접 컨볼루션이다.

[0087] 기존 모바일 및 임베디드 DNN 프레임워크(예: DarkneTZ, TensorFlow Lite)는 im2col(Image to Column) 컨볼루션을 사용하여 시간이 많이 걸리는 컨볼루션 레이어를 수행한다. im2col은 다차원의 데이터를 행렬로 변환하여 행렬 연산을 하도록 해주는 함수를 말한다. 다차원 데이터의 컨볼루션은 im2col를 통해 행렬로 변환된 데이터의 내적과 같다. Im2col 컨볼루션은 도 8a와 같이 각 패치(즉, 요소가 필터의 요소에 매핑되는 입력 액티베이션 집합)를 2차원 행렬로 평면화하고 평면화된 패치와 필터 간의 행렬 곱셈을 수행하여 빠른 컨볼루션 레이어 실행을 달성할 수 있다. 하지만 패치를 병합하면 병합된 패치를 저장하기 위한 추가 버퍼를 할당해야 하므로 컨벌루션 레이어의 작업 세트 크기가 크게 증가한다. SRAM의 제한된 용량(최대 수백 KB)으로 인해 작업 세트 크기가 증가하면 DNN 실행 속도가 크게 느려지는 느린 DRAM 액세스가 많이 발생한다. 따라서 빠른 DNN 실행을 달성하기 위해 컨볼루션 레이어의 작업 세트 크기를 최소화해야 한다.

[0088] 컨볼루션 레이어의 작업 세트 크기를 최소화하기 위해 본 발명에서는 im2col 컨볼루션 대신 직접 컨볼루션을 사용할 수 있다. 직접 컨볼루션은 입력에 컨볼루션 레이어의 필터를 상관 연산하고 필터를 슬라이딩 윈도우 방식으로 모든 영역에서 상관값을 계산한 결과값이 출력이 된다. 직접 컨볼루션은 도 8b와 같이 필요한 입력 액티베이션을 요청 시 가져오기 때문에 작업 세트 크기를 늘리지 않는다. 감소된 작업 세트 크기 외에도 직접 컨볼루션은 입력 액티베이션의 고유한 시간 및 공간적 지역성으로 인해 SRAM 데이터의 재사용을 증가시키는 경향이 있다. 직접 컨볼루션이 입력 액티베이션에 걸쳐 필터를 공간적으로 슬라이드하여 출력 액티베이션의 채널을 생성하기 때문에 최근에 접근한 입력 액티베이션에 인접한 입력 액티베이션이 가까운 미래에 접근될 가능성이 높다. 필터의 경우 출력 채널을 생성할 때 동일한 필터를 여러 번 사용하기 때문에 필터는 시간적 지역성이 높다. 수요 페이징 내장 SRAM과 결합된 직접 컨볼루션의 높은 공간적 및 시간적 집약성은 느린 DRAM 액세스를 크게 줄이는 데 도움이 될 수 있다.

[0089] 도 9a 및 9b는 DNN 친화적인 SRAM 관리 과정을 설명하는 도면이다.

[0090] 도 9a에 보여진 바와 같이, ARM 트러스트존 기술이 적용된 기존 오픈 소스 TEE의 프레임워크인 Pager는 요구 페이징(Demand Paging)을 사용하여 보안 임베디드 SRAM의 제한된 용량을 관리한다. 요구 페이징은 필요할 때마다 암호화된 DRAM에서 SRAM으로 데이터를 로드하고 가장 최근에 사용되지 않은 SRAM 데이터를 DRAM으로 푸시하여 SRAM 공간을 회수한다. 요구 페이징은 데이터의 시간적 지역성을 효율적으로 활용하지만 컨볼루션 레이어의 입력 액티베이션, 필터 및 출력 액티베이션의 다양한 접근 패턴을 인식하지 못한다. 또한 SRAM의 크기는 모바일

및 임베디드 장치에서 수백 KB에 불과하므로 컨벌루션 레이어에 필요한 모든 데이터를 SRAM에 저장할 수는 없다. 이러한 비효율적인 SRAM 관리는 느린 DRAM 접근 횟수를 크게 증가시키고 빠른 DNN 실행을 방해한다. 따라서 SRAM이 다수의 느린 DRAM 액세스를 유발하지 않도록 컨볼루션 레이어의 다양한 데이터 접근 패턴을 통합하여 작은 SRAM을 효율적으로 관리해야 한다.

[0091] 도 9b에 보여진 바와 같이, 본 발명은 SRAM 데이터의 재사용을 최대화하기 위해 DNN 친화적인 SRAM 관리를 구현할 수 있다. 여기에서, DNN 친화적인 SRAM 관리는 컨볼루션 레이어의 입력 액티베이션, 필터 및 출력 액티베이션의 다양한 접근 패턴을 활용한다. 본 발명은 컨볼루션 레이어 실행 전체에서 반복적으로 접근할 때 먼저 모든 입력 액티베이션을 SRAM에 고정한다. 출력 채널을 생성하기 위해 접근되는 입력 액티베이션은 다른 모든 출력 채널에 대해 다시 접근되어야 한다. 그런 다음 나머지 SRAM 공간을 두 개의 순환큐로 구성한다. 하나는 필터용이고 다른 하나는 출력 액티베이션용이다. 필터는 입력 액티베이션과 같이 자주 접근하는 데이터이다. 그러나 크기가 입력 액티베이션보다 훨씬 작고 하나의 출력 채널만 생성하는 데 필터가 필요하기 때문에 필터를 순환큐에 저장하는 것만으로도 높은 시간적 지역성을 충분히 활용할 수 있다. 출력 액티베이션은 공간적 집약성이 높은 1회 쓰기 데이터이므로 순환 큐에 적합한다. DNN 친화적인 방식으로 SRAM을 관리함으로써 SRAM 데이터의 재사용을 최대화하고 느린 DRAM 액세스를 최소화하여 DNN 실행을 가속화할 수 있다.

[0092] 본 발명은 암호 하드웨어를 사용하여 데이터 암호화 및 복호화를 수행하도록 하여 제한된 CPU 리소스를 DNN 실행에 전적으로 사용할 수 있다. 그렇게 함으로써, CPU 리소스를 DNN 실행에 완전히 전용할 뿐만 아니라 암호 하드웨어의 고성능을 활용하여 빠른 DNN 실행을 달성할 수 있다. 데이터 암호화 및 복호화는 SRAM이 암호화된 DRAM에서 데이터를 로드 및 저장하고 제한된 CPU 리소스의 상당량을 소비할 때마다 발생한다. 데이터 암호화 및 복호화의 높은 오버헤드를 극복하기 위해 암호 하드웨어를 사용하여 암호화 및 복호화 작업을 수행할 수 있다. 오버헤드가 높은 데이터 암호화 및 복호화를 암호 하드웨어로 오프로딩하여 제한된 CPU 리소스를 완전히 전용하고 DNN 실행을 가속화할 수 있다(도 10 참조).

[0093] 도 11은 암호 하드웨어의 오프로딩을 설명하는 도면으로, (a)는 인터럽트 기반 오프로딩이고, (b)는 DMA(Direct Memory Access) 지향 오프로딩이다.

[0094] 본 발명은 도 11의 (a) 인터럽트 기반 오프로딩 및 (b) DMA 지향 오프로딩의 두 가지 하드웨어 오프로딩 메커니즘 중에서 선택할 수 있다. 두 오프로딩 메커니즘은 서로 다른 주변장치 사용 시나리오에서 가속화될 수 있다. 인터럽트 기반 오프로딩은 주변장치가 주어진 작업 처리를 완료하자마자 인터럽트를 생성하므로 대기 시간이 중요한 작업을 주변장치에서 수행하는 데 적합한다. 반면에 DMA 지향 오프로딩은 한 번에 더 많은 양의 데이터를 처리하는 데 적합한다. DMA는 CPU가 개입하지 않고 데이터를 처리하기 위한 모든 데이터 전송 및 주변 호출을 처리하고 모든 데이터가 처리된 후 인터럽트를 발생시킨다. DNN 친화적인 SRAM 관리는 SRAM과 DRAM 간에 대량 데이터 전송(일반적으로 수십 KB)을 요구하므로 인터럽트 기반 오프로딩보다 더 큰 페이로드 크기로 더 높은 처리량을 달성하는 DMA 기반 오프로딩이 유리할 수 있다.

[0095] 암호 하드웨어를 사용하면 제한된 CPU 리소스를 DNN 실행에 완전히 전용하여 DNN 실행을 가속화할 수 있고, 추가적인 성능 최적화가 가능하다. CPU와 암호 하드웨어에서 각각 실행되는 중복 DNN 실행 및 데이터 암호화 및 복호화이다. 컨벌루션 레이어의 한 가지 속성은 서로 다른 출력 채널을 생성하는 작업이 서로 독립적이라는 것이다. 컨벌루션 레이어는 하나의 필터를 사용하여 하나의 출력 채널을 생성하고 입력 액티베이션은 모든 출력 채널 간에 읽기 전용 데이터로 공유된다. 이 속성은 풀링 레이어에도 적용된다. 풀링 레이어의 각 출력 채널은 해당 입력 채널만 요구하므로 출력 채널의 작업을 데이터 병렬로 만든다. 이를 기초로 컨볼루션 레이어의 대량 출력 채널 실행을 데이터 복호화 단계, 연산 단계 및 데이터 암호화 단계의 세 가지 파이프라인 단계로 분할할 수 있다. 그런 다음, 출력 채널의 서로 다른 벌크의 세 단계를 파이프라인하여 더 빠른 DNN 실행을 달성할 수 있다. 이를 인트라-레이어 파이프라이닝이라 한다. 이는 컨볼루션 레이어의 다양한 출력 채널 작업을 파이프라인으로 연결하기 때문이다.

[0096] 도 12는 신경망 연산의 수행 과정에서 인트라-레이어 파이프라이닝의 구현을 설명하는 도면으로, 256개의 출력 채널(SRAM의 제한된 용량으로 인해 대량의 32개 출력 채널 포함)을 생성하고 AlexNet의 실행 대기 시간에 가장 크게 기여하는 AlexNet의 6번째 레이어에 인트라-레이어 파이프라이닝을 적용할 때의 이점을 보여준다.

[0097] 도 12에서, 데이터 복호화 단계는 먼저 대량의 출력 채널을 생성하는 데 필요한 입력 액티베이션 및/또는 필터를 복호화하고 SRAM에 로드한다. 그런 다음 연산 단계에서는 입력 액티베이션과 필터를 사용하여 대량의 출력 채널을 연산한다. 그 후 데이터 암호화 단계에서 대량의 출력 채널을 암호화하여 DRAM에 저장한다. 인트라-레이어 파이프라이닝이 없으면 (a)와 같이 3단계가 직렬화되고 레이어를 실행하는 데 57ms가 걸린다. 반면에, 출력

채널의 서로 다른 벌크의 3단계를 파이프라인하는 인트라-레이어 파이프라이닝은 (b)와 같이 3단계가 병렬화되어 레이어를 실행하는 데 (a)의 경우보다 24.6% 더 빠른 43ms밖에 걸리지 않는다. 이 예는 인트라-레이어 파이프라이닝이 더 빠른 DNN 실행을 달성할 수 있음을 보여준다. 도 13은 컨벌루션 레이어에 인트라-레이어 파이프라이닝을 적용하기 위한 의사 코드를 보여주는 알고리즘이다.

[0098] 그러나 컨볼루션 레이어에 인트라-레이어 파이프라이닝을 적용하면 파이프라인을 사용하지 않은 레이어 실행보다 SRAM의 용량이 더 많이 소모된다. 인트라-레이어 파이프라이닝의 기능적 정확성을 보장하려면 더 큰 용량이 필요하다. 대량의 출력 채널의 연산 단계와 다음 대량의 출력 채널의 데이터 복호화 단계를 오버랩핑하려면 두 개의 대량에 대한 SRAM 버퍼가 동시에 할당되어야 한다. 더 큰 SRAM 버퍼가 필요하면 레이어의 출력 채널이 더 많은 수의 벌크로 그룹화되므로 DNN 실행 속도가 느려질 수 있다. 그러나 더 큰 SRAM 버퍼가 필요함에도 불구하고 인트라-레이어 파이프라이닝의 성능 이점이 벌크당 출력 채널 수가 적어 잠재적인 성능 저하를 능가할 수 있다. 따라서 기본적으로 컨볼루션 및 풀링 레이어에 대한 인트라-레이어 파이프라이닝을 액티베이션한다. 그러나 매우 작은 임베디드 SRAM이 장착된 모바일 및 임베디드 장치의 경우 잠재적인 성능 저하를 방지하기 위해 인트라-레이어 파이프라이닝을 비액티베이션할 수 있다.

[0099] 도 14는 보안 임베디드 SRAM 및 암호 하드웨어가 장착된 모바일 및 임베디드 장치 위에 본 발명을 구현하는 데 필요한 소프트웨어 구성 예를 설명하는 도면이다.

[0100] 도 14에서, 가장 널리 사용되는 TEE 구현 중 하나인 오픈 소스 TrustZone 기반 구현인 OP-TEE 위에 본 발명을 구현하였지만, 반드시 이에 한정되지 않으며 모든 TrustZone 기반 TEE 구현 위에 구현될 수 있다. 본 발명의 핵심에는 TEE 내의 주어진 입력에 대한 예측을 생성하기 위해 DNN의 레이어를 실행하는 신뢰할 수 있는 애플리케이션인 GuardiaNN 런타임이 있다. DNN의 실행은 REE에서 시작된다. 먼저 REE는 설명(예: 레이어 수, 레이어당 입력 및 출력 액티베이션 크기), 암호화 입력 데이터 및 DNN의 필터를 TEE 메모리로 전송하고 GuardiaNN 런타임을 호출한다. 그런 다음 GuardiaNN 런타임은 각기 다른 유형의 DNN 레이어를 구현하는 헬퍼 함수를 사용하여 DNN의 각 레이어를 수행한다. 그 후 GuardiaNN 런타임은 DNN을 실행하여 만든 예측을 REE로 다시 반환한다.

[0101] 직접 컨볼루션은 GuardiaNN 런타임 내에서만 구현할 수 있으며, GuardiaNN 런타임이 신뢰할 수 있는 OS(예: OP-TEE)와 상호 작용하여 SRAM을 할당하고 암호 하드웨어 및 DMA를 사용해야 한다. DNN 친화적인 SRAM 관리 및 인트라-레이어 파이프라이닝을 구현하려면 GuardiaNN 런타임이 DRAM으로 교체되지 않는 버퍼를 SRAM에 할당해야 한다. 이를 위해 TEE 메모리 할당을 위한 TEE Internal Core API 함수인 TEE_Malloc()을 확장하여 isSRAM이라는 추가 입력 인수를 사용한다. isSRAM의 값이 true이면 신뢰할 수 있는 OS는 SRAM 버퍼를 할당하고 버퍼가 DRAM으로 스왑 아웃되는 것을 방지한다. 또한 신뢰할 수 있는 OS(예: Pager)의 메모리 관리자를 확장하여 isSRAM을 DRAM에 true로 설정한 상태에서 TEE_Malloc()을 호출하여 할당된 SRAM 버퍼를 교체하는 것을 방지한다. isSRAM의 기본값은 false로 설정되어 isSRAM을 인식하지 못하는 기존의 신뢰할 수 있는 애플리케이션의 기능적 정확성을 보장한다. 예를 들어 TEE_Malloc(1024, hint, true)를 호출하면 DRAM에 제거되지 않는 1KB SRAM 버퍼가 할당된다. 여기서 힌트는 버퍼의 특성에 대한 몇 가지 힌트를 제공한다(예: 0으로 채우기).

[0102] DMA 지향 데이터 암호화 및 복호화 오프로딩의 경우 GuardiaNN 런타임은 신뢰할 수 있는 OS의 DMA 장치 드라이버에서 정의한 사용자 지정 시스템 호출을 호출한다. GuardiaNN 구현은 신뢰할 수 있는 OS를 확장하여 EncryptData() 및 DecryptData()라는 두 가지 사용자 지정 시스템 호출을 제공한다. EncryptData() 시스템 호출은 암호 컨텍스트(암호 유형, 키 크기 등 포함), SRAM 시작 주소, DRAM 시작 주소 및 데이터 크기를 입력으로 사용한다. 그런 다음 시스템 호출은 CPU 캐시를 플러시하여 더티 캐시 라인을 SRAM으로 제거하고, SRAM 시작 주소에서 암호화되지 않은 SRAM 데이터를 읽고, 암호화 컨텍스트 및 암호 하드웨어를 사용하여 데이터를 암호화하고, 암호화된 데이터를 DRAM 시작 주소에서 DRAM으로 저장한다. 유사한 방식으로 DecryptData() 시스템 호출은 암호 컨텍스트, DRAM 시작 주소, SRAM 시작 주소 및 데이터 크기의 네 가지 인수를 입력으로 사용한다. 그런 다음 EncryptData() 시스템 호출과 유사한 절차를 따라 암호화된 DRAM 데이터를 읽고, 데이터를 복호화하고 복호화된 데이터를 SRAM에 저장한다.

[0103] 확장된 TEE_Malloc() API 기능과 기존 GlobalPlatform API와 함께 두 개의 사용자 지정 시스템 호출을 사용하면 본 발명을 모바일 및 임베디드 장치 위에 충실하게 구현할 수 있다.

[0105] 평가(EVALUATION)

[0106] 실험 셋업(Experimental Setup)

[0107] 빠르고 안전한 DNN 실행에 대한 GuardiaNN의 효과를 평가하기 위해 STM32MP157C-DK2 개발 보드 위에 GuardiaNN을 프로토타입으로 만들고 DNN 실행 속도 및 에너지 소비를 기본 보안 DNN 프레임워크와 비교하였다. 개발 보드는 오픈 소스 트러스트존(TrustZone) 기반 TEE 구현인 OP-TEE에서 공식적으로 지원하며 최신 임베디드 장치의 일반적인 하드웨어 구성을 반영하였다. 듀얼 코어 ARM Cortex-A7 CPU, 256KB 보안 임베디드 SRAM, 암호 하드웨어 및 512MB DDR3L DRAM으로 구성된다. 신뢰할 수 있는 OS로서 GlobalPlatform TEE Client API v1.1 및 Internal Core API v1.0을 지원하는 Pager와 함께 OP-TEE v3.11.0을 사용한다. GuardiaNN 및 기본 프레임워크의 구현은 OP-TEE가 현재 단일 TEE 인스턴스 내에서 멀티스레딩을 지원하지 않기 때문에 하나의 CPU 코어만 사용한다. ARM NEON 단일 명령어 다중 데이터 명령어로 DarkneTZ의 DNN 레이어 구현을 확장하여 각 DNN 레이어 작업 간의 데이터 병렬성을 활용하였다. 확장 DNN 레이어 구현은 GuardiaNN과 기본 프레임워크 모두에 적용된다. 신뢰할 수 있는 OS가 모든 SRAM(TEE와 REE 사이의 공유 메모리로 OP-TEE가 예약한 4KB 제외)을 GuardiaNN과 기본 프레임워크 모두에 할당한다고 가정한다. 에너지 소비 비교를 위해 Monsoon HVPM(High Voltage Power Monitor)를 사용하고 장치 전체의 에너지 소비를 측정한다.

[0108] 벤치마크로서 8비트 정수 양자화를 사용하여 양자화된 8개의 DNN을 선택하고 민감한 사용자 및 DNN 데이터와 관련된 5개의 대표적인 모바일 및 임베디드 애플리케이션 도메인을 다룬다. 5개 도메인은 이미지 분류, 얼굴 인식, 지문 인식, 시선 추적 및 감정 인식이다. 각 도메인에 대해 다양한 DNN이 존재한다. 그러나 여기에서 실행하기에 합리적으로 짧은 실행 대기 시간을 가진 각 도메인에서 대표적인 경량 DNN을 선택한다. 예를 들어 STM32MP157C-DK2 개발 보드에서 기본 프레임워크가 DNN을 실행하는 데 192초가 걸렸기 때문에 이미지 분류를 위한 DNN인 ResNet-18을 벤치마크로 포함하지 않는다. 하기 표 2에는 8개의 DNN과 그 특성이 나열되어 있다.

[0109] [표 2]

[0110]

Domain	Name	# Layers	Input Size
Image classification	AlexNet [35]	11	3x32x32
	MobileNetV1 [25]	29	3x32x32
	Tiny Darknet [63]	21	3x32x32
	VGG-7 [37]	11	3x32x32
Face recognition	DeepID [69]	9	3x31x31
Fingerprint recognition	FgptAuth [66]	11	1x128x128
Gaze tracking	GAZEL [58]	10	1x64x64
Emotion recognition	Smart Doll [11]	11	3x50x50

[0111] 빠른 DNN 실행(Fast DNN Execution)

[0112] 먼저 선택한 모든 DNN의 실행 대기 시간을 측정하여 GuardiaNN의 DNN 실행 속도를 평가한다. 제안된 기술의 기여도를 분석하기 위해 기본 프레임워크에서 시작하여 각 제안된 기술을 점진적으로 적용하여 DNN 실행 대기 시간을 측정한다. 직접 컨볼루션 및 DNN 친화적인 SRAM 관리, 암호 하드웨어 및 인트라-레이어 파이프라인에 대한 작업을 하면 도 15에 5개의 막대로 표시된 총 5개의 구성이 제공된다. 실험에서 인트라-레이어 파이프라이닝을 위해 8개 출력 채널의 벌크 크기를 사용한다.

[0113] 도 15의 (a)는 대기 시간 실험 결과를 보여준다. 대부분의 DNN 실행은 기본 프레임워크에서 엄청나게 느려 최대 11.4초의 대기 시간이 발생한다. GuardiaNN은 모든 DNN에서 대기 시간을 1초 미만으로 줄인다. 도 15의 (b)는 직접 컨볼루션과 DNN 친화적인 SRAM 관리 기술(세 번째 막대)이 적용된 구성으로 정규화된 GuardiaNN이 제공하는 상대적 속도 향상을 보여준다. 각각의 제안된 기술이 효과적임을 관찰할 수 있다. 직접 컨벌루션을 적용하면 im2col을 사용하는 기본 프레임워크에 비해 기하 평균 속도가 2.58배 향상된다. DNN 친화적인 SRAM 관리는 3.19배의 추가 기하 평균 속도 향상을 가져온다. 암호화 및 복호화를 암호 하드웨어로 오프로딩하면 1.73배의 추가 기하 평균 속도 향상이 제공된다. 인트라-레이어 파이프라이닝을 적용하면 이를 더욱 개선하여 기하 평균 속도를 1.07배 높일 수 있다. GuardiaNN은 베이스라인에 비해 15.3배의 기하 평균 속도 향상을 달성한다. 여기서 평가하는 8개의 DNN 중에서 GuardiaNN은 Smart Doll을 가장 가속화하여 베이스라인보다 31.4배 더 빠르게 한다. 이러한 속도 향상의 대부분은 DNN 친화적인 SRAM 관리를 직접 컨볼루션과 함께 적용한 결과 느린 DRAM 액세스 수가 감소했기 때문이다. AlexNet은 GuardiaNN의 암호 하드웨어 및 인트라-레이어 파이프라인 사용에서 가장 큰 이점을 얻는다. 레이어 연산에 비해 AlexNet의 암호화 및 복호화 오버헤드는 암호 하드웨어 및 인트라-레이어 파이프라인을 사용하여 크게 줄인 다른 DNN보다 크다. 하기 표 3은 GuardiaNN가 DNN 실행 속도에 미치는 영향을 보여준다.

[0114] [표 3]

Name	Pager	+Direct Conv	+SRAM Mgmt	+Crypt HW	+IntraLayer Pipelining
AlexNet [35]	1.00x	1.28x	2.81x	8.26x	9.92x
DeepID [69]	1.00x	1.12x	7.93x	13.25x	13.80x
FgptAuth [66]	1.00x	2.98x	7.89x	15.54x	16.76x
GAZEL [58]	1.00x	8.68x	20.59x	26.21x	26.54
MobileNetV1 [25]	1.00x	1.59x	3.31x	5.90x	6.54x
Smart Doll [11]	1.00x	8.44x	22.20x	30.58x	31.45x
Tiny Darknet [63]	1.00x	1.10x	4.11x	8.32x	9.35x
VGG-7 [37]	1.00x	3.59x	19.28x	25.26x	25.94x

[0115]

[0116] 상기 표 3에 보여진 바와 같이, 본 발명에서 제안한 GuardiaNN은 보안 보장을 손상시키지 않고 광범위한 DNN의 실행을 가속화할 수 있다.

[0117] 고에너지 효율(High Energy Efficiency)

[0118] 이제 벤치마크에서 각 DNN에 대한 모든 구성에서 DNN 실행의 에너지 소비를 조사한다. 먼저 유휴 상태와 DNN 실행 중에 장치 전체의 평균 전력을 측정하고 DNN 실행으로 인한 평균 전력 증가를 계산하기 위해 두 값을 뺀다. 그런 다음 평균 전력 증가와 대기 시간을 곱하여 DNN 실행의 에너지 소비를 계산한다. 정규화된 결과는 도 15의 (b)에 나와 있다. 제안된 각 기법을 적용함에 따라 에너지 소비가 감소하는 것을 관찰할 수 있다. 기본 프레임워크와 비교하여 GuardiaNN의 에너지 소비는 기하 평균 92.3% 감소하여 에너지 효율성이 15.2배 향상되었다. 이는 GuardiaNN이 제공하는 상당한 대기 시간 감소 때문일 수 있다. 제안된 기술을 적용한 후에도 DNN 실행 중에 장치 전체 전력은 동일한 수준으로 유지된다. 감소된 대기 시간과 결합된 GuardiaNN은 기본 프레임워크보다 훨씬 더 높은 에너지 효율성을 달성한다.

[0120] 민감도 연구(Sensitivity Studies)

[0121] 인트라-레이어 파이프라이닝의 벌크 크기가 DNN 실행 속도에 미치는 영향을 연구하기 위해 4개 벌크 크기(4, 8, 16 및 32 출력 채널)로 GuardiaNN의 DNN 실행 대기 시간을 측정한다. 도 16은 벌크 크기 4로 정규화된 측정값을 보여준다. 4개의 벌크 크기 외에도 각 DNN을 최저 대기 시간으로 이끄는 최적의 레이어별 벌크 크기 집합을 계산하고 다음과 같이 대기 시간을 측정하였다. 인트라-레이어 파이프라이닝이 최적의 벌크 크기로 대기시간을 최대 14.24% 향상시키는 것을 관찰할 수 있다(도 16의 다섯 번째 막대). 대기 시간이 가장 높은 벌크 크기와 비교된다. 대부분의 DNN의 경우 대량 크기가 증가함에 따라 대기 시간이 감소하는 경향이 있다. 벌크 크기가 클수록 암호 하드웨어에 대한 암호화 및 복호화 요청(및 인터럽트) 수가 감소하기 때문이다. 이는 파이프라인 연산 단계의 대기 시간을 효과적으로 줄여 전체 대기 시간을 줄이다. 그러나 벌크 크기는 SRAM 용량에 구속된다. 예를 들어 FgptAuth에 8보다 큰 벌크 크기를 사용하면 필요한 메모리 크기가 SRAM 크기를 초과한다. 이것이 벌크 크기가 16 및 32인 FgptAuth의 실행 대기 시간이 도 16에서 비어 있는 이유이다.

[0122] GuardiaNN에서 사용하는 DMA 가능 암호 하드웨어는 여러 블록 암호 및 작업 모드를 지원한다. GuardiaNN은 보안 강화를 위해 AES를 사용하며 여기서는 AES-ECB와 AES-CBC의 두 가지 작동 모드를 비교한다. (a) 암호 하드웨어 및 (b) OP-TEE의 기본 CPU 기반 암호화 라이브러리인 LibTomCrypt가 있는 CPU에서 실행하여 다양한 키 크기로 AES-ECB 및 AES-CBC 암호화 및 복호화의 처리량을 측정한다. 도 17은 측정된 처리량을 보여준다. LibTomCrypt를 사용하는 CPU에서보다 암호 하드웨어에서 AES-ECB 및 AES-CBC가 훨씬 빠르게 실행되는 것을 관찰하였다. 또한 키 크기가 길수록 AES 암호화 및 복호화에서 더 많은 라운드가 발생하므로 LibTom-Crypt를 사용하는 CPU의 처리량은 키 크기가 증가함에 따라 감소한다. 작동 모드를 비교하면 AES-CBC는 체인으로 인해 AES-ECB보다 처리량이 약간 적다. 반대로 암호 하드웨어에서는 키 크기나 작동 모드가 처리량에 눈에 띄는 영향을 미치지 않는다. 이는 암호화 및 복호화 성능이 암호 하드웨어 및 DMA 버퍼 관리 비용에 의해 제한되기 때문이다. 그래도 GuardiaNN에서 사용하는 암호 하드웨어는 모든 경우에 CPU보다 훨씬 더 높은 암호화 및 복호화 효율성을 제공한다.

[0123] 모바일 및 임베디드 기기에서 DNN 실행에 대한 관심이 높아지면서 해당 기기에서 DNN 실행을 가속화하는 다양한 기술이 등장하였다. 그러나 연합 학습과 같은 장치 내에서 처리되는 데이터는 차등 프라이버시를 기반으로 하기 때문에 여전히 많은 프라이버시 문제가 있다. 따라서 DNN에 트러스트 실행 환경을 활용하는 것이 합리적인 방향

이다. 예를 들어, SecureTF는 트러스트 실행 환경을 활용하는 TensorFlow 기반의 분산 보안 기계 학습 프레임워크이다. PPFL은 로컬 교육 및 집계, 다중 파트 ML에 트러스트 실행 환경을 활용하여 안전한 연합 학습을 가속화한다. Chiron 및 Myelin은 기계 학습 내에서 트러스트 실행 환경을 서비스로 활성화한다.

[0124] DarkneTZ, Infenclave 및 Slalom은 모두 트러스트 실행 환경 내에서 DNN의 일부를 실행하도록 제안한다. 그러나 나머지 레이어를 공격에 노출시키고 순진하게 접근 방식을 적용하면 상당한 양의 성능 오버헤드가 발생한다. 이러한 문제를 해결하기 위해 HybridTEE는 원격 서버의 트러스트 실행 환경에 DNN 실행을 요청하는 것을 제안한다. 하지만 HybridTEE의 가속 효과는 로컬 트러스트 실행 환경의 DNN 실행을 최적화하지 않기 때문에 그다지 중요하지 않다.

[0125] 본 발명은 로컬 트러스트 실행 환경에서 DNN의 완전한 보호를 제안할 뿐만 아니라 실제 모바일 및 임베디드 환경에서 실행할 수 있도록 하는 놀라운 속도 향상을 제공할 수 있다.

[0126] DNN 암호화(DNN Encryption)

[0127] DNN을 보호하는 또 다른 방향은 DNN 데이터를 암호화하는 것입니다. 예를 들어, SecureML은 확장 가능한 개인 정보 보호 DNN 프레임워크를 구축하기 위해 안전한 다자간 계산을 활용하였다. SoftME는 신뢰할 수 있는 환경을 제공하고 암호화 및 복호화 및 연산 단계로 구성된 신뢰할 수 있는 작업을 실행한다. SoftME로 DNN을 실행하면 기밀성이 보장되지만 데이터 암호화 및 복호화에 CPU를 사용하고 큰 성능 오버헤드가 발생한다. 그 중 동형암호를 적용하면 암호화된 데이터에 대한 연산이 가능하기 때문에 유망할 수 있다. CryptoNets는 그러한 아이디어가 실현 가능함을 보여주고 아이디어를 보안 교육으로 확장한다. MiniONN은 사전 훈련된 DNN을 인식하지 못하도록 변환하는 기술을 제안한다. 또한 동형 암호화를 활용하여 안전한 연합 전이 학습 프로토콜을 구성한다. 그러나 동형암호는 연산 처리량이 낮고 모바일 및 임베디드 장치에 비실용적인 것으로 간주되는 경우가 많다.

[0128] TEE는 ARM TrustZone 및 Intel SGX가 널리 사용되는 상용 구현으로 높은 보안 보장으로 인해 관심을 끌고 있다. 소프트웨어 기반 보안 솔루션이 적용될 수 있지만, 다양한 애플리케이션을 보호하기 위해 성공적으로 악용되었다. 그러나 캐시 아키텍처, 이중 인스턴스 앱 또는 중첩 앱과 같은 TEE 시스템을 대상으로 하는 많은 위협이 있다. 이에 따라 보안 강화를 위한 최근 몇 가지 제안이 있다. 또한 TEE 활용의 어려움을 완화하기 위한 많은 작업이 제안되었다. 최소 커널은 TEE의 제한된 메모리 문제를 해결하기 위해 작은 커널을 구축한다. CoSMIX는 애플리케이션 수준의 보안 페이지 오류 처리기를 허용한다. TEEMon은 TEE를 위한 성능 모니터링 프레임워크이다.

[0130] 본 발명에 따른 트러스트 환경 기반 인공지능 장치는 트러스트 실행 환경에서 인공신경망 실행을 격리하고 동작 속도가 느린 DRAM에 저장된 데이터를 암호화하여 보안을 강화할 수 있다. 또한, 직접 컨볼루션 및 SRAM 관리를 통해 DRAM 접근 횟수를 줄일 수 있고 암호 하드웨어로 데이터 암호화 및 복호화를 오프로딩하고 파이프라이닝을 구현하여 신경망 연산의 수행을 데이터 암호화 및 복호화 연산들과 오버랩핑되게 수행하여 인공신경망 실행을 가속화할 수 있다.

[0132] 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

[0134] 100: 트러스트 환경 기반 인공지능 장치

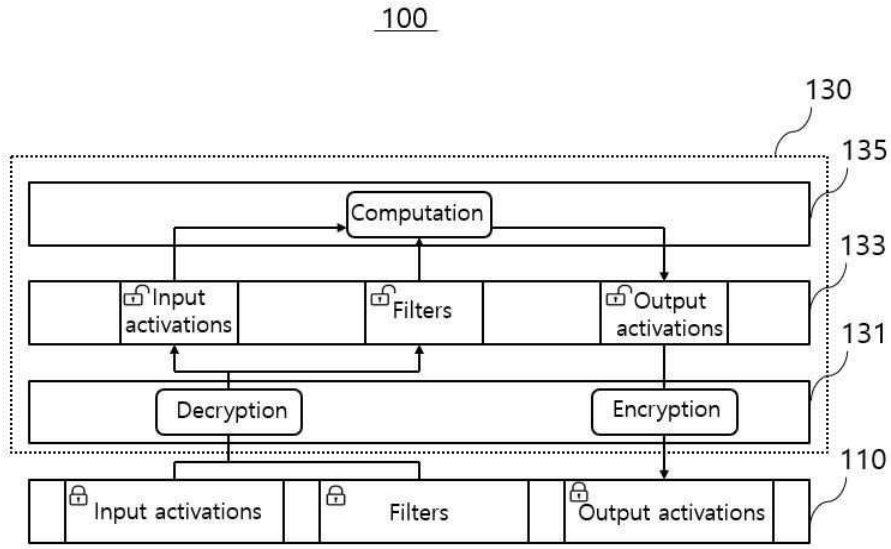
110: 제1 유형의 메모리 130: 트러스트 인공지능 처리부

131: 암호처리 프론트-엔드 프로세스

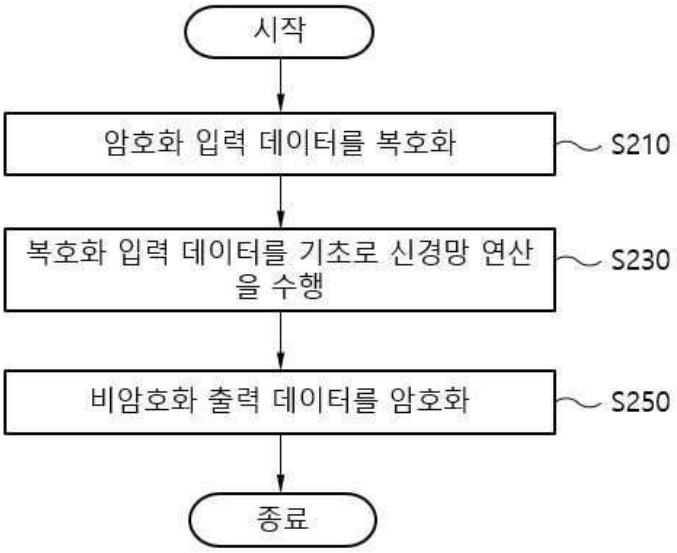
133: 제2 유형의 메모리 135: 프로세서

도면

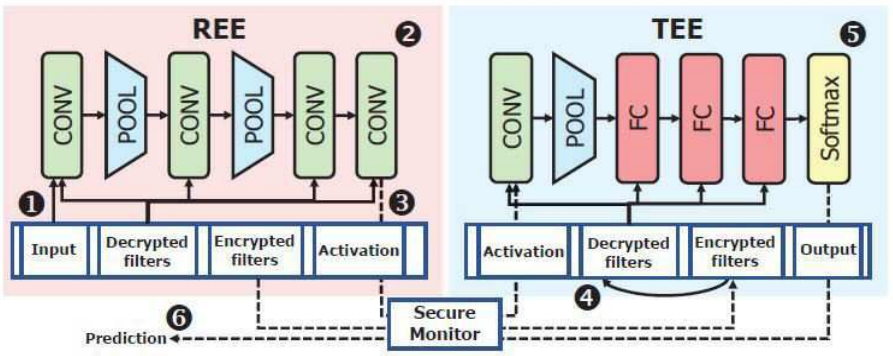
도면1


100
130
135
Computation
133
Input activations
Filters
Output activations
131
Decryption
Encryption
110
Input activations
Filters
Output activations

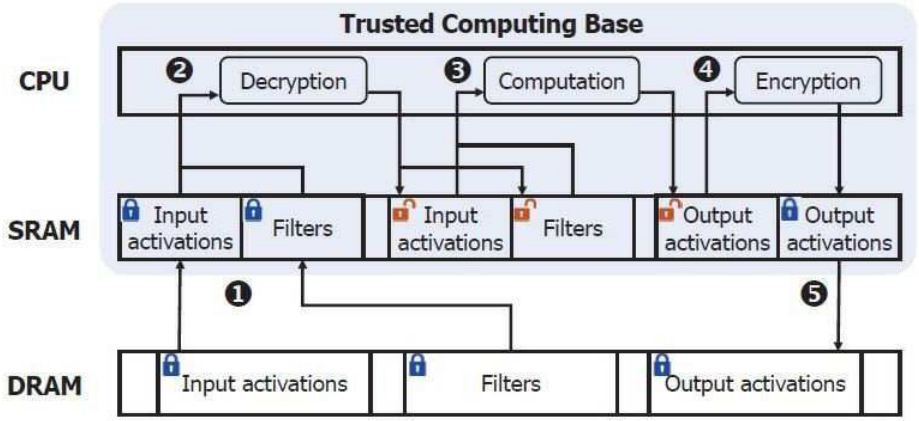
도면2


시작
암호화 입력 데이터를 복호화
S210
복호화 입력 데이터를 기초로 신경망 연산을 수행
S230
비암호화 출력 데이터를 암호화
S250
종료

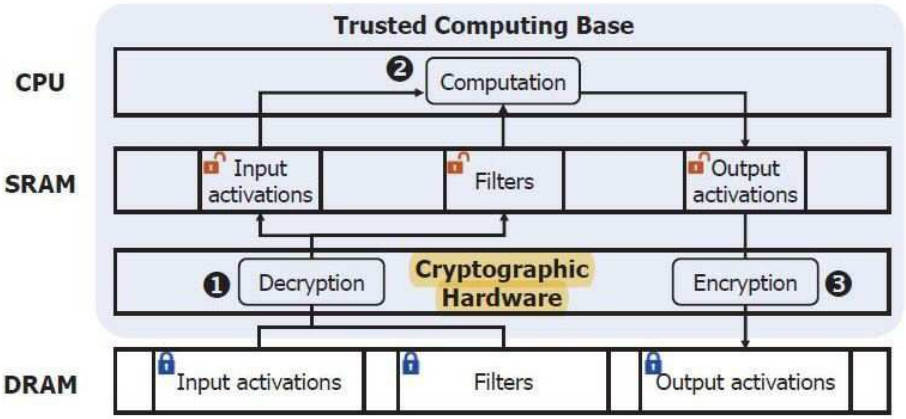
도면3



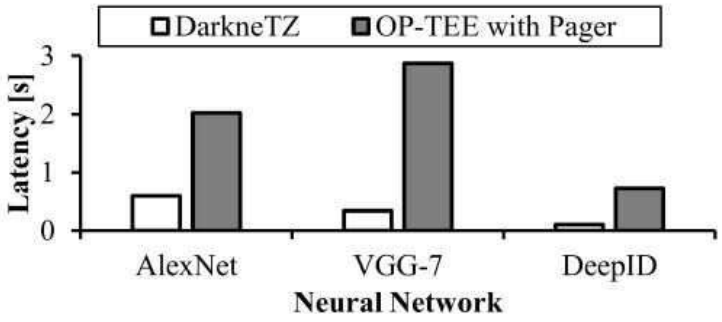
도면4a



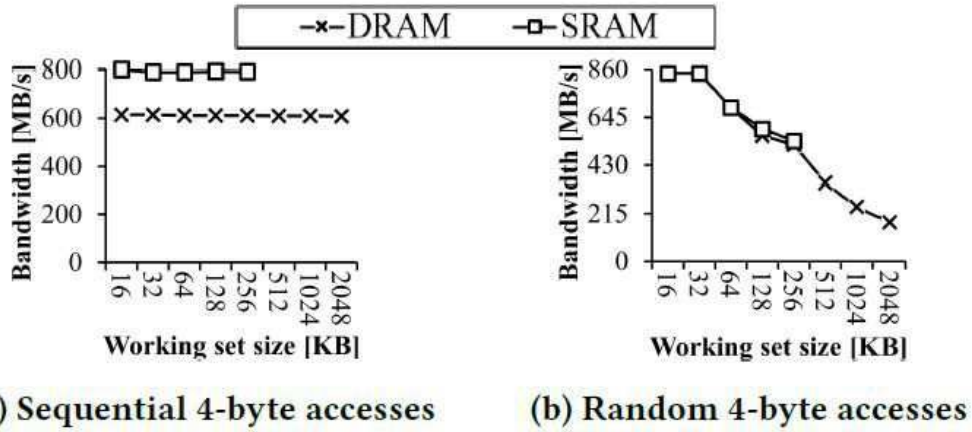
도면4b



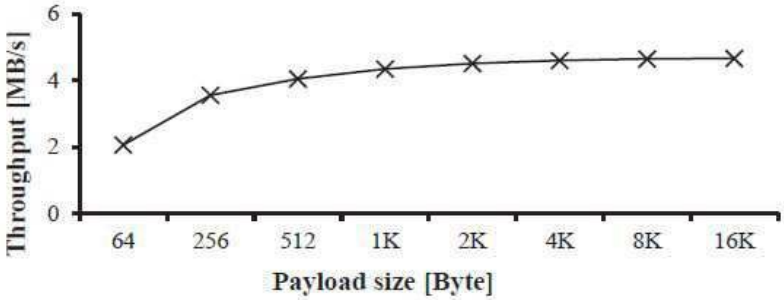
도면5



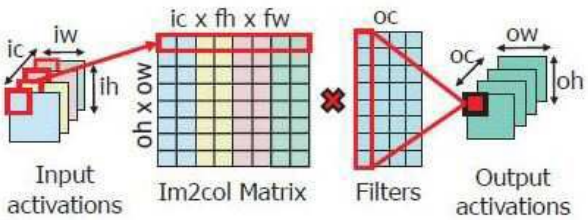
도면6



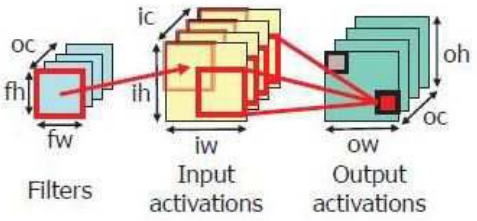
도면7



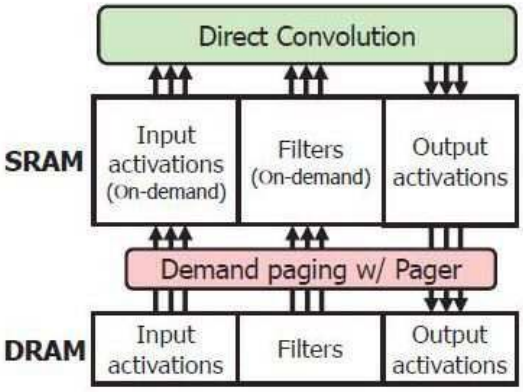
도면8a



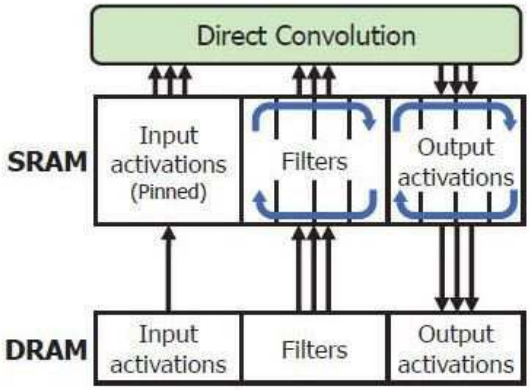
도면8b



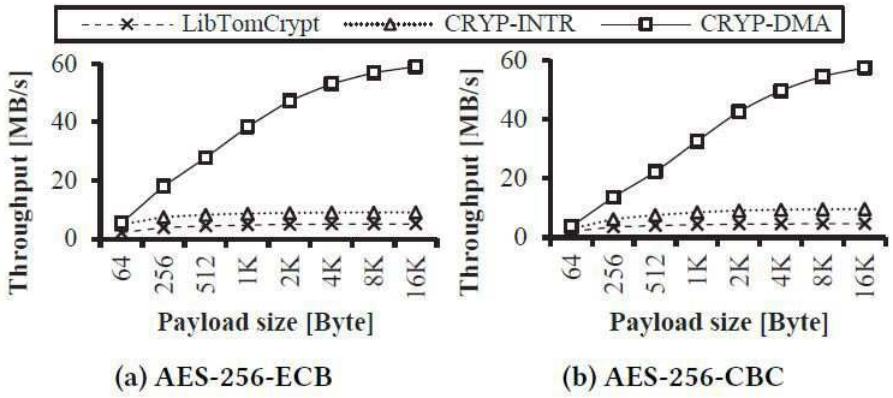
도면9a



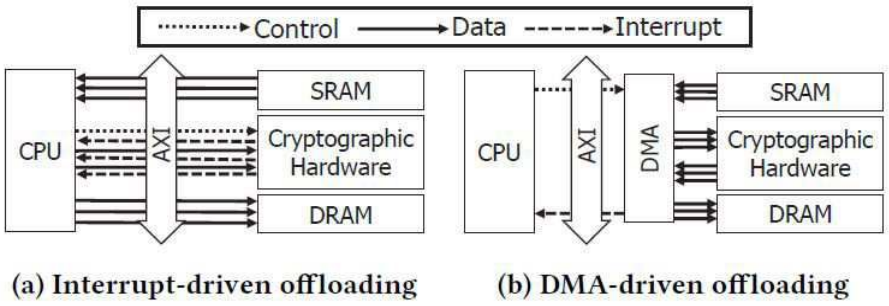
도면9b



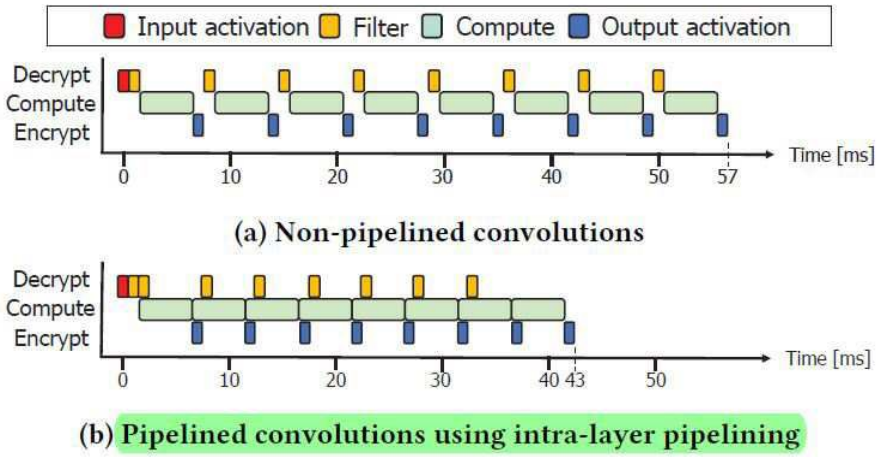
도면10



도면11



도면12



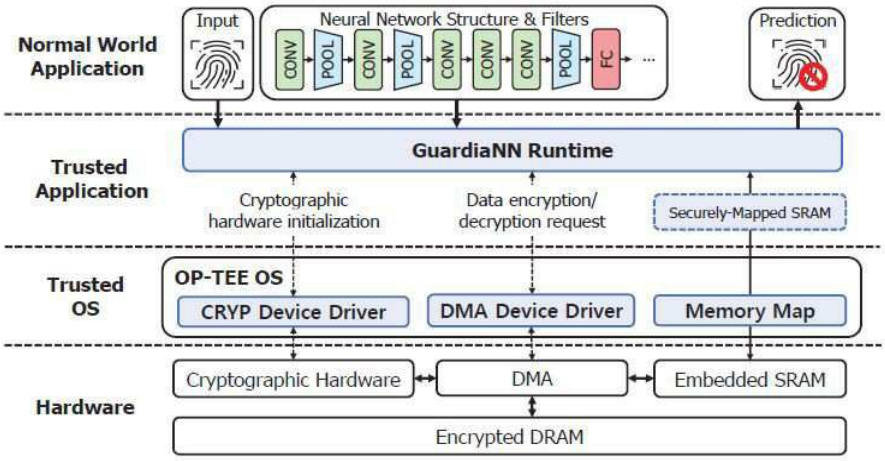
도면13

Algorithm 1 Intra-layer pipelining for convolutional layers

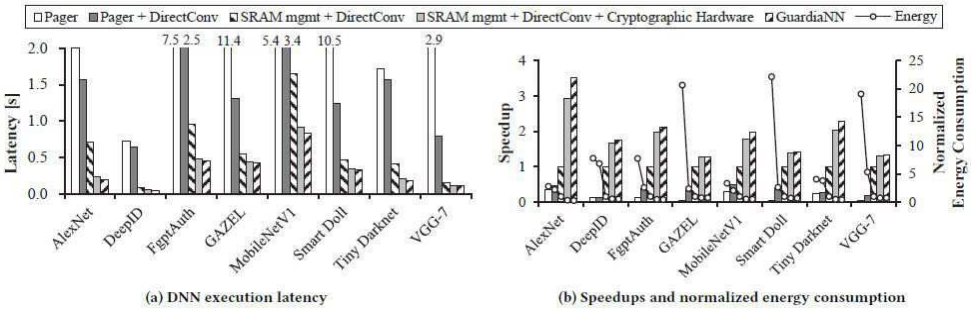
D_{OC}: The first output channel for each intra-layer pipelining stage
$CrypBlkSize$: Cryptographic hardware invocation granularity
$D_{filters,oact}$: DRAM addresses for filters and output activations; CHW format
S_{iact}: SRAM address for input activations; CHW format
$S_{filters,oact}$: In-SRAM circular queues for filters and output activations
f, o: # filters / # output activations of a bulk of output channels
F, O: # filters / # output activations to be requested to cryptographic hardware
$crypCtx$: the cryptographic context for the convolutional layer

procedure PIPELINEDCONVLAYER
  $F = \lceil f/CrypBlkSize \rceil * CrypBlkSize$
  $DecryptData(crypCtx, D_{filter}, S_{filter}.tail, F)$
  $S_{filter}.tail += F$
  for oc in D_{OC} **do**
    if $(S_{filter}.tail - S_{filter}.head) == 0$ **then**
      $F = \lceil f/CrypBlkSize \rceil * CrypBlkSize$
    else
      $leftover = (S_{filter}.tail - S_{filter}.head - f)$
      $F = \lceil (f + leftover)/CrypBlkSize \rceil * CrypBlkSize$
    end if
    $DecryptData(crypCtx, D_{filter}[oc], S_{filter}.tail, F)$
    $S_{filter}.tail = S_{filter}.tail + F$
    $DirectConv(S_{iact}, S_{filter}.head, S_{oact}.tail)$
    $S_{filter}.head = S_{filter}.head + f$
    $S_{oact}.tail = S_{oact}.tail + o$
    $O = \lfloor (S_{oact}.tail - S_{oact}.head)/CrypBlkSize \rfloor * CrypBlkSize$
    $EncryptData(crypCtx, S_{oact}.head, D_{oact}[oc], O)$
    $S_{oact}.head = S_{oact}.head + O$
  end for
end procedure

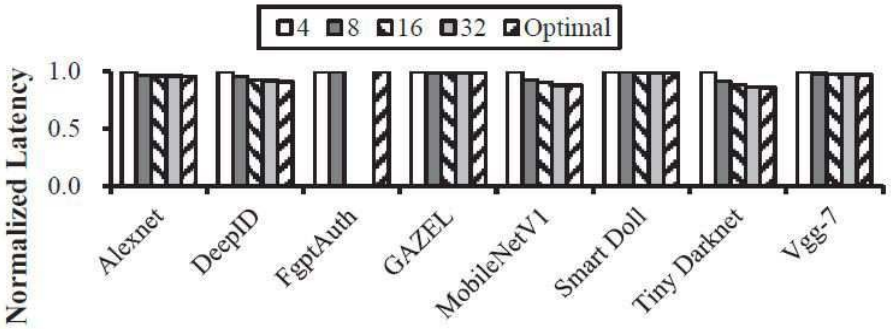
도면14



도면15



도면16



도면17