마이크로컨트롤러를 위한 정적 레이어 퓨전 및 파이프라인 최적화

서승찬, 김영진

아주대학교

chan7431@ajou.ac.kr, youngkim@ajou.ac.kr

Static Layer Fusion and Pipeline Optimization for Microcontrollers

Seung Chan Seo, Young-Jin Kim

Ajou Univ.

요 약

최근 사물인터넷(IoT) 디바이스에서 직접 인공지능을 수행하는 TinyML 기술이 주목받고 있다. 그러나 낮은 클럭 속도와 제한된 연산 능력을 가진 마이크로컨트롤러(MCU)에서 심층 신경망(CNN)을 실행할 때, TFLM에서 제공하는 TinyML 모델의 수행 엔진 백엔드 라이브러리인 CMSIS-NN에서 계층별(Layer-by-Layer) 실행 방식은 빈번한 데이터 로드/스토어(Load/Store)를 유발하여 추론 지연(Latency)의 주원인이 될 수 있다. 이를 해결하기 위해 레이어 퓨전(Layer Fusion) 기법이 연구되고 있으나, 퓨전된 파이프라인 내부에서 발생하는 복잡한 주소 계산과 조건문 분기는 오히려 연산 성능을 저하시키는 병목의 원인이 될 수 있다. 본 논문에서는 Cortex-M4F 기반의 MCU(Arduino Nano 33 BLE) 환경에서 추론 속도를 극대화하기 위해 롤링 버퍼(Rolling Buffer) 기반의 정적 레이어 퓨전을 구현하고, 이에 따른 주소 연산 오버헤드를 제거하기 위한 포인터 호이스팅(Pointer Hoisting) 및 1x2 픽셀 재사용(Pixel Reuse) 기법을 제안한다. 제안하는 구조는 퓨전 루프 내부의 불필요한 Modulo 연산과 주소 곱셈을 제거하고, 각종 최적화를 통해 기존 CMSIS-NN 기반 방식 대비 약 7%정도의 추론 속도 향상을 달성함을 확인하였다.

Ⅰ. 서 론

현재 웨어러블 기기 등 엣지 디바이스에서 데이터를 클라우드로 전송하지 않고 즉각 처리하는 TinyML 수요가 급증하고 있다. 그러나 Arduino Nano 33 BLE와 같은 소형 기기는 64MHz 수준의 낮은 연산 능력을 가지고 있어, 실시간성을 요구하는 딥러닝 모델을 구동하는 데 어려움이 있다. TinyML로 널리 사용되는 딥러닝 framework는 TFLM(TensorFlow Lite for Microcontrollers)로 수행 엔진의 백엔드 라이브러리는 자체 라이브러리 외에 최적화된 C 라이브러리인 CMSIS-NN[1]을 채택하고 있다.

TFLM의 일반적인 딥러닝 수행 엔진은 대상 딥러닝 모델의 한 레이어의 연산을 모두 마친 후 결과를 메모리에 기록하고, 다음 레이어에서 다시 읽어오는 방식을 사용한다. 이러한 방식은 데이터의 지역성(Locality)을 저하시키고 불필요한 메모리 접근 사이클을 소모하여 전체 추론 시간을 증가시킨다. 이를 개선하기 위해 연산자를 결합하는 레이어 퓨전(Layer Fusion)이 대안으로 제시되지만, 퓨전된 레이어 간의 데이터 흐름을 제어하기 위한 인덱싱 연산(Indexing Overhead)이 추가되어 순수 연산 효율이 떨어지는 문제가 발생한다.

본 논문에서는 이러한 문제를 해결하기 위해 TFLM의 수행엔진 처리 방식에 대해 두 개의 Conv2D 레이어를 융합하고, 중간 데이터를 효율적으로 전달하는 최적화된 수행엔진을 설계하였다. 구체적으로, 데이터 접근 시 발생하는 반복적인 주소 연산 비용을 제거하기 위해 포인터 호이스팅(Pointer Hoisting) 기법을 고안하였으며, 인접한 픽셀의 데이터를 레지스터 단계에서 공유하는 1x2 픽셀 재사용(Pixel Reuse) 기법을 도입하여 메모리 로드 횟수를 절감하고 DSP 연산 효율을 극대화하였다.

Ⅱ. 본 론

1. 제안하는 정적 레이어 퓨전 구조

본 연구에서는 연속된 두 개의 Conv2D 레이어(Conv1, Conv2)의 수행 방식을 분석하고 병목이 되는 부분을 파악하고 이를 개선하도록 하여 하나의 연산 블록으로 융합(Fusion)하여 데이터 전송 효율을 높였다. Conv1은 전체 출력 맵을 한 번에 생성하지 않고, Conv2가 연산을 수행하기 위해 필요한 최소한의 행(Row) 단위로 데이터를 생성하여 롤링 버퍼(Rolling Buffer)에 전달한다. 전달이 완료되면 롤링 버퍼가 받은 데이터로 Conv2는 연산을 시작한다.

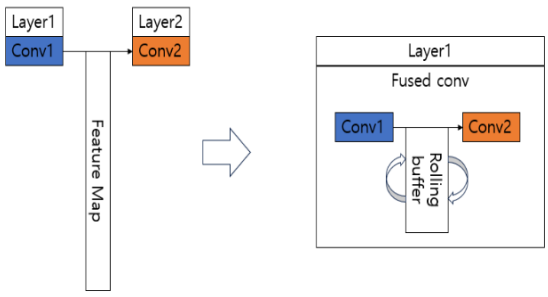

그림1. 기존의 계층별 실행 방식(좌)과 제안하는 롤링 버퍼 기반 정적 레이어 퓨전 방식(우)의 메모리 구조 비교

제안하는 방식은 전체 피처맵 대신 커널 연산에 필요한 최소한의 행만 유지하여 메모리 사용량을 최소화하며 매번 할 필요가 없던 주소 연산의 횟수를 줄여 연산 오버헤드를 감소했다[2]. 이러한 구조는 데이터가 레지스터에 머무는 시간을 늘려주지만, 버퍼를 순환하며 접근해야 하는 구조적 특성상 복잡한 인덱싱을 필요로 한다. 본 연구는 이 인덱싱 오버헤드를

최적화하는 데 집중하였다.

2. 포인터 호이스팅(Pointer Hoisting) 최적화

롤링 버퍼는 원형 큐(Circular Queue) 구조를 가지므로, 데이터 접근 시 매번 (y % Buffer_Height)와 같은 Modulo(나머지) 연산이 필요하다. 수백만 번 반복되는 합성곱 연산의 내부 루프(Inner Loop)에서, 나눗셈을 포함하는 Modulo 연산은 심각한 성능 저하를 일으키는 요인이다.

본 연구에서는 이를 해결하기 위해 다음과 같이 포인터 호이스팅 기법을 적용하였다.

- 기존 방식: 루프 내부에서 매 픽셀마다 Address = Base + ((y % H) * W + x) * C를 계산하여 접근.
- 제안 방식: 내부 루프 진입 전, y축(행)에 대한 물리적 메모리 주소 계산을루프 밖으로 끌어올려 포인터 배열(row_ptrs)에 저장(Hoisting).

이를 통해 가장 빈번하게 실행되는 내부 루프에서는 고비용의 Modulo 연산과 곱셈 연산을 완전히 제거하고, 단순 포인터 덧셈(row_ptrs[ky] + offset)만으로 데이터를 로드할 수 있어 연산 속도를 비약적으로 향상시켰다 .

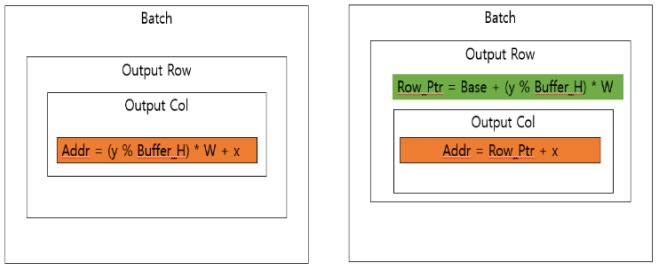

그림2. 포인터 호이스팅 기법 적용 전후의 주소 연산 흐름 비교

고비용의 Modulo, Multiplication 연산 및 주소 계산을 가장 빈번한 내부 루프(Inner Loop)에서 상위 루프로 이동(Hoisting)시켜 연산 효율을 극대화하였다.

3. 1x2 픽셀 재사용

Cortex-M4F의 DSP 명령어 효율을 높이기 위해 1x2 픽셀 재사용(Pixel Reuse) 기법을 적용하였다. 인접한 두 개의 가로 픽셀을 동시에 처리함으로써 가중치(Weight)를 레지스터에 한 번 로드하여 두 번 재사용한다. 이는 메모리 로드 명령어(Load Instruction)의 실행 횟수를 50% 절감하는 효과가 있다. 또한, __SMLAD(Signed Multiply Accumulate Dual) 명령어를 사용하여 한 사이클에 두 번의 곱셈-누적 연산을 수행하여 연산 밀도를 극대화하였다[1].

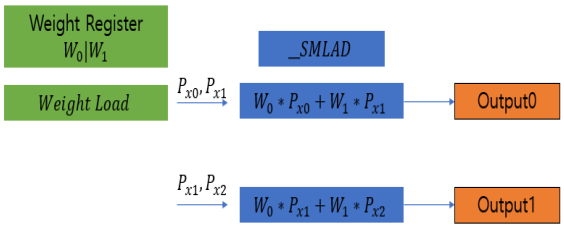

그림3. 1x2 픽셀 재사용 및 SIMD 명령어 활용 구조

가중치(Weight)를 레지스터에 상주시키고 인접한 2개의 입력 픽셀에 대해 재사용함으로써 메모리 접근 횟수를 50% 절감하였다.

표 1. 기존 방식과 제안 기법의 MNIST 적용 결과

모델	기존방식(us)	제안방식(us)	변화율(%)
MNIST1	59625	55372	7.1
MNIST2	77569	71642	7.6

Ⅲ. 결 론

본 연구에서 제안한 커널의 속도 향상을 평가하기 위해 Arduino Nano 33 BLE 보드 상에서 실험을 진행하였다. 실험 모델은 3x3 커널을 사용하는 표준 CNN 레이어 2개인 MNIST dataset[3]의 수행 모델을 대상으로 하였다.

표1에서 보는 바와 같이 실험 결과를 분석해보면, 제안하는 커널이 기존 방식 대비 평균 약 7.3%의 추론 개선을 이뤘다. 이러한 성능 향상의 주된 원인은 다음 두 가지로 요약할 수 있다.

첫째, 포인터 호이스팅(Pointer Hoisting)의 효과이다. 기존 방식은 롤링 버퍼의 원형 큐 구조로 인해 가장 빈번하게 호출되는 내부 루프(Inner Loop)에서 매 픽셀마다 Modulo 연산(%)과 곱셈 연산을 수행해야 했다. 제안하는 방식은 이를 루프 진입 전 미리 계산된 포인터 배열(row_ptrs)을 참조하는 방식으로 변경함으로써, 내부 루프의 연산을 획기적으로 줄였다.

둘째, 1x2 픽셀 재사용에 따른 메모리 병목 완화이다. 인접한 두 픽셀을 동시에 처리하는 루프 언롤링을 통해 가중치(Weight) 로드 횟수를 절반으로 줄이고, Cortex-M4F의 SIMD 명령어(__SMLAD) 활용도를 높여 연산 밀도를 극대화한 것이 전체적인 실행 시간 단축에 기여하였다.

결론적으로, 이 실험은 제한된 연산 자원을 가진 MCU 환경에서 단순한 Layer Fusion은 연산 오버헤드 문제를 야기하고, 이를 구조적으로 해결하였다는 점에서 의의가 있다.

ACKNOWLEDGMENT

이 논문은 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원-대학ICT연구센터(ITRC)의 지원을 받아 수행된 연구이며 (IITP-2026-RS-2021-II212051) 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(RS-2025-23523557).

참 고 문 헌

[1] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs," arXiv preprint arXiv:1801.06601, 2018.

[2] J. Lin, W. M. Chen, H. Cai, C. Gan, and S. Han, "MCUNetV2: Memory-efficient patch-based inference for tiny deep learning," Advancesin Neural Information Processing Systems, vol. 34, 2021.

[3] MNIST dateset, https://www.kaggle.com/datasets/hojjatk/mnist-dataset..