논문 2023-18-13

임베디드 시스템에서의 객체 분류를 위한 TVM기반의 성능 최적화 연구

(TVM-based Performance Optimization for Image Classification in Embedded Systems)

허 청 환, 예 민 해, 신 익 희, 이 대 우*
(Cheonghwan Hur, Minhae Ye, Ikhee Shin, Daewoo Lee)

Abstract : Optimizing the performance of deep neural networks on embedded systems is a challenging task that requires efficient compilers and runtime systems. We propose a TVM-based approach that consists of three steps: quantization, auto-scheduling, and ahead-of-time compilation. Our approach reduces the computational complexity of models without significant loss of accuracy, and generates optimized code for various hardware platforms. We evaluate our approach on three representative CNNs using ImageNet Dataset on the NVIDIA Jetson AGX Xavier board and show that it outperforms baseline methods in terms of processing speed.

Keywords : Embedded System, Code optimization, Artificial Intelligence, Quantization, Deep Learning, TVM

Ⅰ. 서 론

최근 심층 신경망은 기존의 복잡한 알고리즘으로 접근했던 문제를 간단하게 해결하는 강력한 방법론으로 떠오르고 있다. 이는 객체 인식 [1], 얼굴 인식 [2], 음성처리 [3] 등과 같은 작업을 수행하는 분야에 급진적인 변화를 가져왔다. 또한, 모바일과 사물인터넷 환경에서도 최첨단 기능을 제공하기 위해 활발하게 사용되고 있다. 하지만, 기존 딥러닝 접근법은 계산 집약적이며 모델이 작동하기 위해 많은 양의 자원을 요구한다 [4]. 결과적으로, 임베디드 장치에 대한 심층 추론을 수행하는 것은 긴 실행 시간과 CPU, GPU, 메모리 및 전력을 포함한 엄청난 양의 자원의 소비로 인해 매우 어려운 문제이다. 이를 해결하기 위해 임베디드 장치에 대한 심층 추론을 최적화하는 수많은 접근법이 제안되었다.

본 논문에서는 객체 분류를 위한 심층 신경망을 임베디드 시스템에 맞게 최적화하기 위하여 TVM (Tensor Virtual Machine) [5]을 이용하는 방안을 제안한다. TVM은 대표적인 딥러닝 컴파일러 중 하나로, 심층 신경망 모델 연산을 타겟 임베디드 디바이스의 실행코드로 변환하는 작업을 수행한다. TVM 커뮤니티가 활성화됨에 따라 TVM에 다양한 기능이 추가되고 있는데, 본 논문에서는 심층 신경망 자체를 경량화하는 연산 최적화, 심층 신경망 연산을 수행하는 코드를 최적화하는 코드 최적화, 심층 신경망이 탑재된 임베디드 소프트웨어 개발 및 평가를 간소화하는 응용 최적화를 융합하는 방안을 제안한다.

대표적인 심층 신경망 최적화 기법의 하나는 가중치를 양자화 [6]하는 것이다. 심층 신경망 대부분에서 가중치와 활성화 함수는 모델의 정확도를 높이기 위해 32 bit floating point (FP32)로 표현된다. 양자화는 이 bit 수를 줄임으로써 모델의 크기를 줄이는 것을 의미한다. 양자화 접근법은 일반적으로 다음과 같이 3가지 기준으로 나누어진다: 1) 학습 중 [7]/학습 후 [8], 2) 동적 [9]/정적 [10], 3) 양자화 정도 [11] (16bit, 8bit, 4bit …). 하지만 대부분 양자화 기술은 샘플 데이터를 요구하기 때문에, 샘플 데이터를 추가로 구할 수 없는 연구 분야에서는 활용하기 어렵다. 본 논문에서는 샘플 데이터 없이 수행할 수 있는 TVM의 Automatic Mixed Precision (AMP) 양자화 기법 [12]을 심층 신경망 모델 경량화에 적용한다 [13]. AMP 양자화 기법은 추가적인 학습 과정 없이 경량화 시 정확도 손실 비율에 따른 모듈 중요도를 나눠 컬러링을 함으로써 FP32의 가중치 표현을 빠르게 FP16으로 표현할 수 있다.

심층 신경망 모델 구조는 위와 같은 기법을 통해 경량화할 수 있다. 그러나 한정된 자원을 갖는 임베디드 시스템의 경우 메모리와 처리능력에 대한 효율성을 극대화하는 것이 가장 중요한 문제이기 때문에, 심층 신경망 모델 연산을 수행하는 코드를 최적화하는 과정이 필요하다. 하지만, 하나의 연산을 표현할 수 있는 코드의 개수는 매우 많으며, 타겟 디바이스의 특성에 따라 최적의 코드가 달라지므로, 최적의 코드를 선정하는 것은 어려운 작업이다. 이를 해결하기 위하여 본 논문에서는 TVM의 Auto-Scheduler [14] 기반 코드 최적화를 활용한다 [15]. Auto-Scheduler는 동적 프로파일링과 기계 학습 모델을 기반으로 자동으로 코드 후보들에 대한 실제 실행 시간을 추정하고 이를 통해 타겟 디바이스

*Corresponding Author (dwlee@rtst.co.kr)
Received: Mar. 23, 2023, Revised: May 1, 2023, Accepted: May 18, 2023.
C. Hur, M. Ye, I. Shin: RTST (Senior Researcher)
D. Lee: RTST (Principal Researcher)
※ 본 논문은 2021년도 정부 (과학기술정보통신부)의 재원으로 '자율주행 기술개발혁신사업'의 지원을 받아 수행된 연구임 (No.2021-0-00905, (3세부) Cloud, Edge, Car 3-Tier 연계 인지/판단/제어 SW 및 공통 SW 플랫폼 기술 개발).

에 적합한 최적의 코드를 선별함으로써, 다양한 하드웨어에 적합한 심층 신경망 모델 연산을 도출할 수 있다.

임베디드 시스템에 최적화된 심층 신경망 모델의 코드를 생성했음에도 불구하고, 이를 실제 시스템에 탑재하여 응용 프로그램과 연동시키는 과정은 간단하지 않을 수 있다. 심층 신경망 라이브러리는 대부분 범용으로 개발되었기 때문에 필요하지 않은 기능까지 포함되어 복잡도가 높다. 또한, 코드 비공개 및 사용권 등의 이유로 코드 분석이 불가능하거나 사용 자체가 불가능할 수 있다. 본 논문에서는 이러한 경우를 해결하기 위해, TVM의 Ahead-Of-Time (AOT) 컴파일을 통해 심층 신경망 모델의 연산을 수행하는 C 코드 생성 방법을 소개한다 [16]. 생성된 C 코드를 응용과 함께 빌드함으로써 추론에 필수적인 기능만 최적화하여 타겟 시스템에 탑재할 수 있고, TVM 런타임 모듈을 지원하지 않은 환경에서도 인공지능 응용 프로그램을 쉽게 개발할 수 있다.

TVM 기반의 최적화 성능 연구를 평가하기 위해 NVIDIA Jetson AGX Xavier 보드 [17]에서 대표적인 CNN인 ResNet50 [18], MobileNetV2 [19], EfficientNetV2-S [20]를 사용하였고, ImageNet Dataset [21] 의 이미지 하나를 50회 처리한 시간의 평균값으로 처리속도를, ImageNet 1k Dataset으로 정확도를 평가한다. 평가 방법으로는 TVM기반의 Auto-Scheduler과 AMP에 대한 각각의 성능을 보여주고, TensorRT [22]의 성능과 다양한 조건에서 비교한다. 마지막으로 AOT 컴파일의 적용 예시를 보여준다.

본 논문의 주요 기여는 다음과 같다:

- 추가적인 샘플 데이터 없이 심층 신경망 구조를 경량화함으로써 임베디드 디바이스에 탑재를 가능하게 하도록 TVM의 AMP 기반 연산 최적화 방안을 제안한다.
- 최소한의 사용자 개입만으로 다양한 하드웨어에 최적화된 코드를 생성하기 위하여 TVM의 Auto-Scheduler 기반 코드 최적화 방안을 제안한다.
- 추론에 필수적인 기능만 타겟 시스템에 탑재함으로써 심층 신경망 처리 과정이 포함된 임베디드 소프트웨어 개발을 간소화하기 위하여, TVM의 AOT 컴파일 기반 응용 최적화 방안을 제안한다.

Ⅱ. 관련 연구

1. 양자화

가중치 양자화는 심층 신경망 모델에서 가중치의 정밀도를 낮추고 모델의 계산 복잡성과 메모리 요구사항을 줄이는 기법이다. 가중치 양자화에서 가중치 값은 더 작은 수의 비트, 일반적으로 32비트에서 16비트, 8비트 이하로 반올림된다. 대표적인 가중치 양자화 방법 3가지는 다음과 같다. 고정 소수점 양자화 [23]는 가중치는 정수 또는 고정 소수점 숫자로 표시되며 소수 부분에 대한 비트 수를 강제로 제한함으로써 수행된다. 2의 거듭제곱 양자화 [24]는 2의 거듭제곱인 정수로 표시하여 시프트 및 더하기를 사용하여 효율적인 하드웨어 구현이 가능하다. 마지막으로, 클러스터링 기반 양자화 [25]는 유사성에 따라 클러스터로 가중치가 그룹화되고 각 클러스터는 단일 값으로 표시된다.

하지만 위와 같은 일반적인 양자화 기법은 샘플 데이터를 구하기 어려운 환경에서의 적용이 어렵다. 이러한 문제를 해결하기 위해 본 논문에서는 입력된 심층 신경망 모델을 구성하는 연산을 FP16으로 양자화하는 AMP를 사용한 최적화를 수행한다.

2. 코드 최적화

코드 최적화는 일반적으로 컴파일러에서 생성되는 코드의 성능과 효율성을 개선하는 프로세스를 의미한다. 대표적으로 루프 최적화, 명령 스케줄링, 프로파일링 최적화와 같은 연구가 있다. 루프 최적화는 코드 내 루프를 펼치고, 벡터화 및 병렬화하여 루프의 성능을 향상하게 한다. [26]은 LLVM의 루프 벡터화와 같은 해당 프로세스를 자동화하는 알고리즘을 제안했다. 명령 스케줄링은 파이프라인 중단을 최소화하고 리소스 활용도를 개선하기 위해 명령 순서를 재정렬한다. [27]은 GRIP과 같은 도구를 활용하여 스케줄링을 자동화하는 알고리즘을 제안했다. 프로파일링 최적화는 프로파일링 도구를 사용해 응용 프로그램의 성능 병목 현상을 식별한 후 타겟 최적화를 수행한다. [28]는 Intel VTune 및 NVIDIA Nsight와 같은 다양한 도구와 캐시 최적화 및 코드 재구성 기법을 개발했다.

하지만 기존의 연구들은 다양한 측면에서의 코드 최적화를 수행하기 위해 하이퍼 파라미터를 실험적으로 지정한다. 이러한 문제점을 해결하기 위해 본 논문에서는 TVM에서 제공되는 Auto-Scheduler 기반의 코드 최적화를 사용한다.

3. TVM

TVM은 대표적인 딥러닝 컴파일러 중 하나로, 입력된 심층 신경망 모델 연산을 타겟 임베디드 디바이스에 실행코드로 변환하는 작업을 수행한다. 따라서, TensorFlow [29], PyTorch [30], ONNX [31]와 같은 심층 신경망 프레임워크와 CPU, GPU, FPGA 등 다양한 구조의 디바이스를 지원한다.

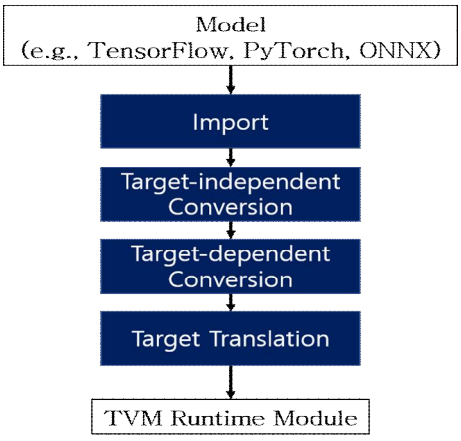

그림 1. TVM 실행 과정
Fig. 1. TVM Execution Process

TVM의 실행 과정은 그림 1에서와 같이 크게 4단계로 구성된다. Import 단계에서는 입력된 심층 신경망을 Relay라는 중간 표현으로 변환한다. 타겟 독립 변환 단계에서는 연산자 퓨전 (operator fusion), 데이터 레이아웃 변경 등 디바이스 특성이 필요하지 않은 최적화가 진행된다. 타겟 종속 변환 단계에서는 임베디드 디바이스에 대해 심층 신경망 모델 연산의 프로파일링 정보를 기반으로 디바이스 특성이 필요한 실행코드 최적화가 진행된다. 마지막으로, 타겟 번역 단계에서 TVM은 지금까지 최적화된 결과를 실제 디바이스 실행코드인 TVM 런타임 모듈을 생성한다. 런타임 모듈은 TVM 런타임 라이브러리와의 연동을 통해 타겟 디바이스에서 실행될 수 있다.

Ⅲ. 제안 기법

1. 임베디드 시스템에서의 객체 분류를 위한 TVM 기반 성능 최적화 기법

그림 2은 임베디드 시스템에서의 객체 분류를 위해 본 논문에서 제안하는 TVM기반 성능 최적화 기법의 순서도이다. 연산 최적화 단계에서는 추론 가속화를 위하여 AMP 기반의 양자화를 통해 심층 신경망 모델이 압축된다. 코드 최적화 단계에서는 Auto-Scheduler를 통해 심층 신경망 전체가 가장 효율적인 코드로 변환된다. 응용 최적화 단계에서는 AOT 컴파일을 통해 연산 구현 정보와 연산 실행 워크플로우 정보가 모두 포함된 C 코드가 생성된다. 생성된 C 코드를 응용과 함께 빌드하면 응용에서 필수적인 기능만 포함된 심층 신경망 처리가 가능해진다.

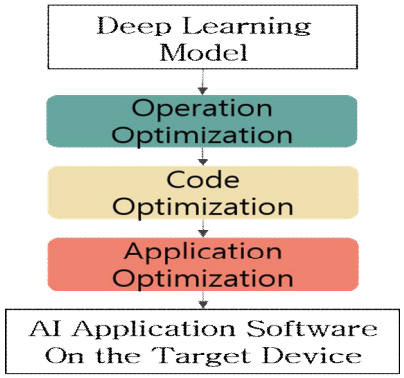

그림 2. 임베디드 시스템에서의 객체 분류를 위한 TVM기반 성능 최적화 순서도
Fig. 2. TVM-Based Performance Optimization Flowchart for Image Classification in Embedded Systems

2. 연산 최적화

연산 최적화를 위한 TVM의 AMP는 심층 신경망 모델에서 가중치와 활성화의 정밀도를 자동으로 최적화하여 성능과 효율성을 향상시킨다. AMP는 반 정밀 (FP16) 및 단일정밀 (FP32) 산술을 조합하여 작동하여 메모리 사용량을 최소화하고 수치적 불안정성과 정확도 손실을 방지한다. 또한, 샘플 데이터 또는 재학습이 필요하지 않기 때문에 빠른 속도로 양자화를 수행할 수 있다.

AMP는 2단계로 구성된다. 첫 번째, 컬러링 단계는 ALLOW, FOLLOW, DENY 세 가지 유형으로 연산을 나누어 색으로 그룹화한다. ALLOW는 정확도 손실이 적고 성능 이득이 높은 연산 (예: MatMul, Conv2d), FOLLOW는 정확도 손실은 적으나 성능 이득도 크지 않은 연산 (예: Add, ReLU), DENY는 정확도 손실이 큰 연산 (예: Exp, Sum)으로 나뉜다. 그림 3은 AMP 양자화 과정을 보여준다. 그림 3의 (1)은 초기 FP32 연산 그래프이다. 그림 3의 (2)는 연산의 특성에 따라 색으로 분류하는 컬러링 과정이다. 그림 3의 (3)은 AMP의 마지막 단계인 각 색상의 양자화를 보여준다. 분류된 연산에 따라 FP32 또는 FP16 변환을 추가한 상태이다. 각 연산이 컬러링되면 TVM의 양자화 도구를 사용하여 각 색상에 대해 개별적으로 양자화를 수행한다.

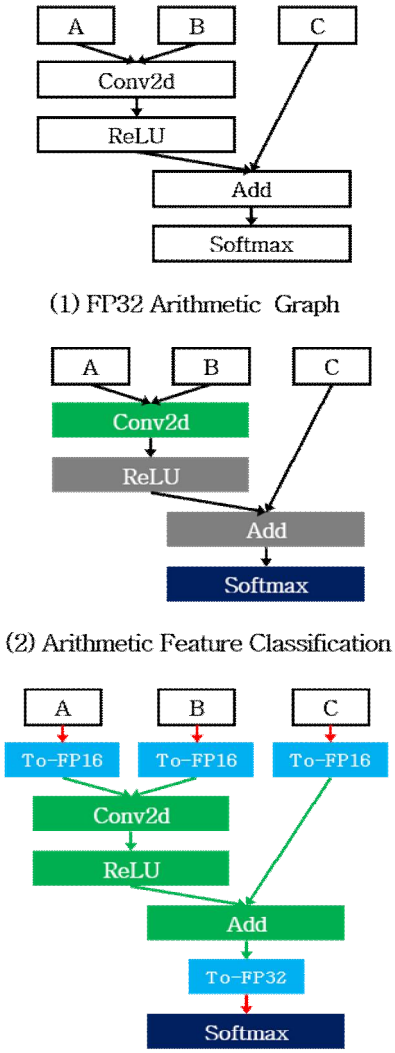

그림 3. AMP 양자화 과정
Fig. 3. AMP Quantization Process

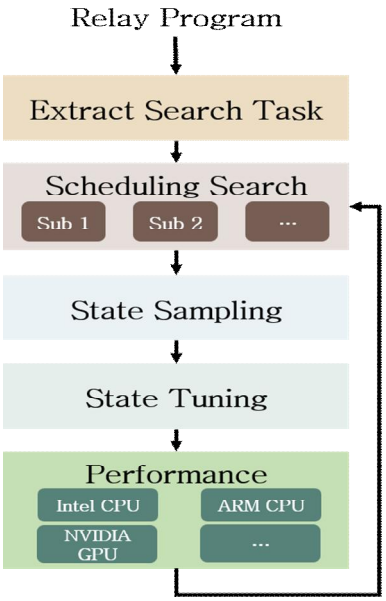

그림 4. Auto-Scheduler 실행 과정
Fig. 4. Auto-Scheduler Execution Process

3. 코드 최적화

코드 최적화를 위한 TVM의 Auto-Scheduler의 실행 과정은 그림 4와 같다. 먼저, Auto-Scheduler는 Relay로 표현된 프로그램을 분석하여 검색 태스크를 추출한다. 여기서 검색 태스크는 신경망 일부분에 대한 최적화 작업을 의미하며, 보통 하나의 검색 태스크는 하나의 심층 신경망 계층을 담당한다. 이후 최적화 과정은 검색 태스크 별로 반복되는데, 검색 태스크 스케줄링 단계에서는 어떤 순서로 검색 태스크를 처리할지가 결정된다. 처음에는 모든 검색 태스크가 일정 횟수만큼 수행되지만, 스케줄링이 진행됨에 따라 최적화 성능이 높은 태스크에 스케줄링 우선순위가 높아진다. 스케줄링 최대 횟수는 사용자 지정 가능하며 일반적으로 검색 태스크 수의 800~1000배 정도의 값이 권장된다.

검색 태스크가 스케줄 되면, 먼저 검색 태스크의 상태를 일정 개수만큼 샘플링 한다. 여기서 상태는 검색 태스크가 담당하는 연산에 대한 저수준 기계어 코드를 추상화한 것으로, 하나의 저수준 기계어 코드 형태와 1대1로 대응된다고 가정한다. Auto-Scheduler는 진화 알고리즘을 통해 샘플링된 상태를 다양한 형태로 변환해보며 각각의 상태에 대한 실행 시간을 평가한다. 디바이스에서 엄청난 양의 상태를 평가할 수 없으므로 상태에서 추출된 특징을 기반으로 실행 시간을 실시간으로 평가할 수 있는 기계 학습 모델을 내부적으로 구성한다. 검색 태스크를 1회 수행할 때마다 Auto-Scheduler는 적은 개수의 상태에 대해서만 프로파일링을 수행하고, 그 결과로 다시 기계 학습 모델을 학습시킨다. 위와 같은 모든 과정은 개발자의 개입 없이 진행되기 때문에 추가적인 리소스 필요 없이 최적의 코드를 구성할 수 있다.

4. 응용 최적화

응용 최적화를 위한 TVM의 AOT 컴파일은 심층 신경망 범용 라이브러리로부터 필수적인 기능만 선택하여 타겟 시스템에서 실행 가능한 최적의 C 코드를 생성 및 빌드하는 기능으로, 생성된 C 코드에는 연산 구현 정보와 연산 실행 워크플로우 정보가 포함된다. 컴파일 단계는 코드 생성과 배포 및 실행으로 크게 2단계로 구성된다. 코드 생성 단계에서는 4.2와 4.3 절에서 수행된 최적화 모델을 입력으로 사용한다. 최적화된 계산 커널을 C와 같은 낮은 수준의 타겟 언어로 코드를 생성한다. 실행코드는 임베디드 디바이스에 의해 직접 로드되고 실행될 수 있으며, 추가 컴파일이나 해석의 필요성이 없다. 그 결과, 기존 TVM 런타임 모듈을 제공하지 않은 대부분의 임베디드 디바이스 운영체제 또는 펌웨어 환경에서도 AOT 컴파일기반의 C 코드 생성을 통하여 인공지능 응용 소프트웨어를 빌드할 수 있다.

AOT 컴파일을 기반으로 인공지능 응용 소프트웨어 수행이 불가능한 임베디드 디바이스에서도 이를 가능하게 하였다. 또한, 범용 라이브러리가 아닌 필수적인 기능만 구현되었기 때문에 성능 평가 및 검증, 보안성이 훨씬 뛰어나다.

Ⅳ. 실 험

1. 실험 설정

실험 환경은 NVIDIA Jetson AGX Xavier 보드에 JetPack 4.5.1와 CUDA 10.2를 설치하여 구축하였고, TVM은 0.9.0dev를 기준으로 설치하였다. 벤치마크로는 대표적인 CNN인 ResNet50, MobileNetV2, EfficientNetV2-S를 사용하였다. 각각의 CNN에 대한 정보는 표 1과 같다.

2. 평가 방법

TVM기반 최적화 성능 분석을 위하여, 본 연구에서는 TVM 최적화 과정을 적용하지 않은 경우, TVM의 Relay 수준 최적화만 적용한 경우, 연산 최적화만 적용한 경우, 코드 최적화만 적용한 경우, 연산 최적화와 코드 최적화를 적용한 경우에 대해 실행 시간 성능을 비교하고, 각각의 경우를 TensorRT 7.1.3 (GPU)과 비교 검증을 수행하였다. 실행 시간은 ImageNet의 224×224 크기의 이미지 하나를 50회 처리한 시간의 평균값으로 정의하였다. 또한, 연산 최적화 시 성능대비 정확도 손실에 대한 실험도 수행하였다. 마지막으로 AOT 컴파일을 통해 얻은 MobileNetV2 코드 실행 예시를 보여준다.

표 1. 실험에 사용된 CNN 정보
Table 1. CNN information used in the experiment

Benchmark	#Weights (M)	GFLOPs	ImageNet Top-1 ACC.
ResNet50	25.6	3.8	76.1
MobileNetV2	3.5	0.3	71.9
EfficientNetV2-S	21.5	8.4	83.9

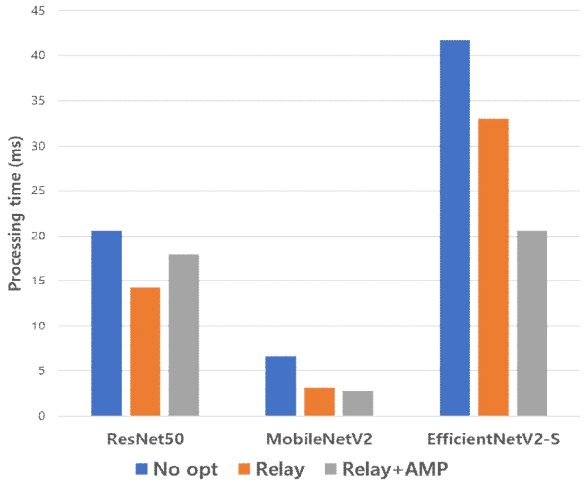

그림 5. AMP 기반 양자화 최적화 성능
Fig. 5. AMP-Based Quantization Optimization Performance

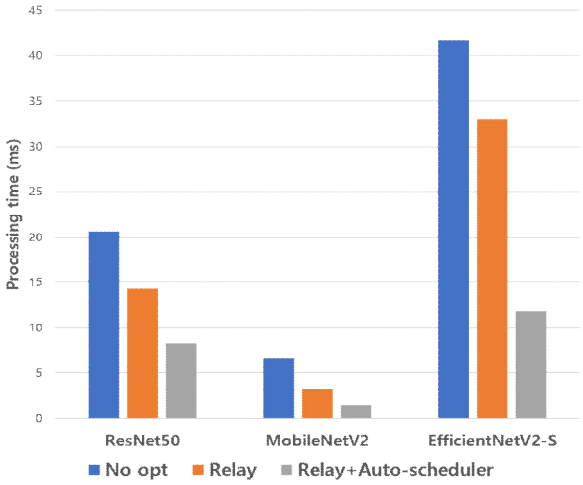

그림 6. 코드 최적화 성능
Fig. 6. Code Optimization Performance

표 2. AMP 연산 최적화에 의한 정확도 성능
Table 2. Accuracy Performance by AMP Computational Optimization

Benchmark	FP32 (Top1)	FP32 (Top5)	FP16 (Top1)	FP16 (Top5)
ResNet50	78.2	94.1	78.0	94.1
MobileNetV2	73.0	91.5	73.0	91.5

3. 연산 및 코드 최적화 성능 분석

그림 5은 AMP 기반 양자화를 사용한 연산 최적화 성능 분석 결과이다. 아무것도 적용하지 않았을 때 대비 Relay 수준 최적화와 AMP를 사용했을 때, 성능이 13~59% 향상되었다. Relay 수준 최적화만 사용한 경우에서 AMP를 추가로 적용한 결과를 비교했을 때 MobileNetV2와 EfficientNetV2-S는 각각 15%, 38%가 향상되었지만, ResNet50의 실행 속도는 25% 감소하였다. ResNet50은 Relay 수준 최적화 시 winograd 알고리즘을 적용하여 행렬 곱셈을 줄일 수 있다. 하지만 AMP를 적용하면 winograd 알고리즘을 지원하지 않아 성능이 감소한다. MobileNetV2와 EfficientNetV2-S는 channel-wise한 구조를 갖고 있으므로 처음부터 winograd 알고리즘을 적용할 수 없어 AMP 과정에서 향상된 최적화 성능을 볼 수 있었다.

표2 [32]는 AMP 기반 양자화를 통한 연산 최적화를 수행하였을 때 정확도 성능으로, ImageNet 1k 데이터에 대한 정확도 손실이 거의 없다. 컬러링을 기반으로 정확도 손실 가능성이 큰 모듈을 제외하고 정확도 손실 가능성은 적지만 압축률을 크게 높일 수 있는 부분만 선택적으로 양자화를 수행하였기 때문에 성능을 유지할 수 있다.

그림 6은 코드 최적화에 대한 성능 측정 결과이다. Auto-Scheduler의 스케줄링 최대 횟수는 20,000번으로 설정하였다. 코드 최적화를 적용한 TVM의 최적화 과정을 통해

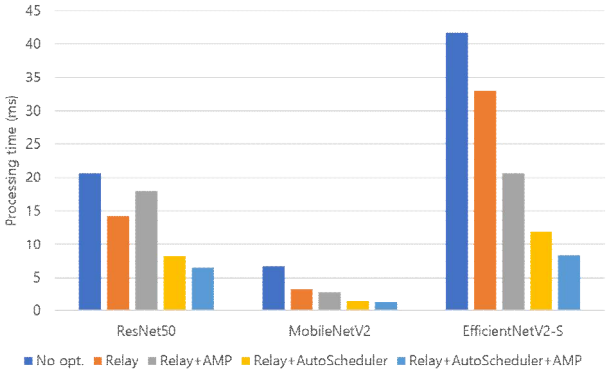

그림 7. Auto-Scheduler와 AMP를 적용한 TVM 최적화 성능
Fig. 7. TVM Optimization with Auto-Scheduler and AMP

CNN의 실행 시간이 60~77%만큼 단축된 것을 확인할 수 있다. 또한, 단축된 실행 시간 중에서 코드 최적화에 의한 비율은 34~71%를 차지한다. 특히, MobileNetV2는 경량화에 비중을 두고 설계된 모델임에도 불구하고, 최적화의 효과가 매우 컸다.

그림 7은 앞서 실험된 연산 최적화와 코드 최적화를 함께 수행하였을 때, 종합적인 성능 평가를 보여준다. 최적화 기술 적용 전 대비 처리속도는 3.2~5.4배 향상되었다. AMP를 적용하지 않은 성능대비 실행 속도는 10~34% 단축되었다. 두 최적화를 동시에 적용하였을 때, 성능 최적화 효율이 크게 향상되었다.

4. 연산 및 코드 최적화 성능 비교

그림 8은 OpenCV (CPU코어 8개) [33], NVIDIA에서 제공하는 NVIDIA GPU용 딥러닝 라이브러리인 TensorRT (GPU)와 TVM의 Auto-Scheduler (GPU)를 비교한 결과이다. OpenCV 대비 TVM은 12~28배 성능향상을 보였고, TensorRT는 11~22배 성능향상을 보여준다. 즉, TVM의 최적화 성능이 좀 더 좋은 것을 알 수 있다. 하지만 ResNet50

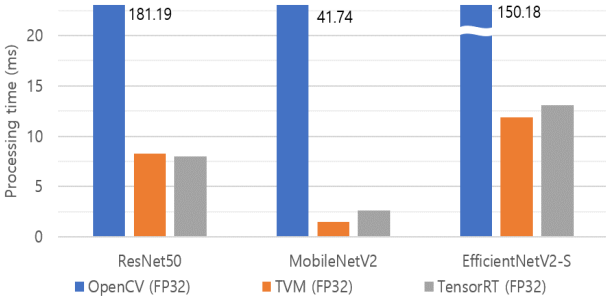

그림 8. OpenCV, Auto-Scheduler, TensorRT 성능 비교
Fig. 8. OpenCV, Auto-Scheduler, TensorRT Performance Comparison

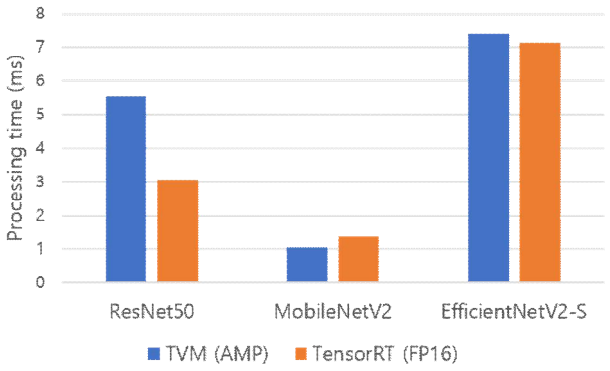

그림 9. Auto-Scheduler, AMP를 적용한 TVM 최적화 성능과 TensorRT (FP16) 비교
Fig. 9. Performance Comparison of TVM Optimization and TensorRT (FP16)

에서는 TVM이 TensorRT보다 0.5ms정도 느린 실행 속도를 보여준다.

그림 9은 Auto-Scheduler과 AMP를 적용한 TVM 최적화 성능과 TensorRT (FP16)을 비교한 그래프이다. 표 3은 그림 9에서 TVM의 성능이 TensorRT와 비슷하다는 것을 수치로 표현한 표이다. 그림 8의 OpenCV와 비교했을 때, TVM 성능의 처리속도가 18.1~34.2배 대폭 향상되었음을 볼 수 있다. TensorRT (FP16)와 비교하였을 때, TVM의 처리속도는 0.48~1.12배 향상으로 느려지는 때도 있었다. 이는 NVIDIA 사에서 자사 보드의 GPU에 최적화된 양자화를 제공하기 때문이라고 분석된다. 하지만 TensorRT는 NVIDIA GPU 상에서만 동작하기 때문에 다양한 하드웨어에 적용하기는 어렵다. TVM 기반의 최적화를 사용하여 NVIDIA GPU를 탑재하지 않은 보드에서 최적화가 가능하므로 TensorRT와 TVM의 성능이 비슷한 것만으로 유의미한 결과이다.

5. 응용 최적화 적용 예시

AOT 컴파일기반 응용 최적화는 TVM 런타임 모듈을 지원하지 않는 환경인 NXP SBC-S32V234 보드 [34]에 당사

표 3. Auto-Scheduler, AMP를 적용한 TVM 최적화 성능과 TensorRT (FP16) 비교 2
Table 3. Performance Comparison 2 of TVM Optimization and TensorRT (FP16)

Method	ResNet50	MobileNetV2	EfficientNetV2-S
TVM (AMP)	5.57	1.04	7.42
TensorRT (FP16)	3.05	1.37	7.13

Input Image

AI Application Software Output

그림 10. 입력 이미지와 AOT 컴파일의 RTWORKS 적용 예시
Fig. 10. Example of AOT compilation for RTWORKS

의 실시간 운영체제인 RTWORKS [35]를 탑재하여 수행하였다. RTWORKS에는 아직 TVM 런타임 모듈이 완전히 포팅되지 않은 상태이다. 그림 10은 AOT 컴파일을 통해 생성된 MobileNetV2의 C 코드를 응용과 함께 빌드하고 이를 수행한 출력 화면이다. 이처럼 AOT 컴파일을 이용하면 TVM 런타임 모듈에 대한 종속성 문제를 해결할 수 있다.

V. 결 론

본 연구에서는 임베디드 시스템에서의 객체 분류를 위하여 TVM 기반의 연산 최적화, 코드 최적화, 응용 최적화라는 3단계 최적화 단계를 제안하고 성능을 분석하였다. NVIDIA Jetson AGX Xavier 보드 상에서 세 가지 CNN 모델을 대상으로 연산 최적화를 적용한 결과, 아무것도 적용하지 않은 경우 대비 13~59%의 성능향상이 있었고, 같은 환경에서 코드 최적화는 60~77% 성능향상을 보여주었다. 연산 최적화와 코드 최적화를 동시에 적용하였을 때 성능이 3.2~5.4배로 대폭 향상되었다. 마지막으로 응용 최적화를 적용하여 TVM 런타임 모듈을 지원하지 않는 임베디드 환경에서도 인공지능 응용 소프트웨어 개발이 가능함을 보여주었다. 향후, NVIDIA GPU 보드가 아닌 다양한 보드에서 TVM 기반의 최적화 성능을 분석할 계획이다.

References

[1] R. Padilla, S. L. Netto, E. A. Da Silva, "A Survey on Performance Metrics for Object-detection Algorithms," 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), pp. 237-242, 2020.

[2] A. Kumar, A. Kaur, M. Kumar, "Face Detection Techniques: A Review," Artificial Intelligence Review Vol. 52, pp. 927-948, 2019.

[3] Y. Kang, Z. Cai, C. W. Tan, Q. Huang, H. Liu, "Natural Language Processing (NLP) in Management Research: A Literature Review," Journal of Management Analytics Vol. 7, No. 2, pp. 139-172, 2020.

[4] J ,Chen, X. Ran, "Deep Learning with Edge Computing: A Review," Proceedings of the IEEE Vol. 107, No. 8, pp. 1655-1674, 2019.

[5] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, "TVM: An Automated End-to-end Optimizing Compiler for Deep Learning," arXiv preprint arXiv:1802.04799, 2018.

[6] H. Wu, P. Judd, X. Zhang, M. Isaev, P. Micikevicius, "Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation," arXiv preprint arXiv:2004.09602, 2020.

[7] M. A. C. Fernandes, H. T. Kung, "A Novel Training Strategy for Deep Learning Model Compression Applied to Viral Classifications," 2021 International Joint Conference on Neural Networks (IJCNN), pp. 1-9, 2021.

[8] Y. Zhou, X. Hu, L. Wang, G. Zhou, S. Duan, "QuantBayes: Weight Optimization for Memristive Neural Networks Via Quantization-aware Bayesian Inference," IEEE Transactions on Circuits and Systems I: Regular Papers Vol. 68, No. 12, pp. 4851-4861, 2021.

[9] N. Shoghi, A. Bersatti, M. Qureshi, H. Kim, "SmaQ: Smart Quantization for DNN Training by Exploiting Value Clustering," IEEE Computer Architecture Letters Vol. 20, No. 2, pp. 126-129, 2021.

[10] B. Liberatori, C. A. Mami, G. Santacatterina, M. Zullich, F. A. Pellegrino, "YOLO-Based Face Mask Detection on Low-End Devices Using Pruning and Quantization," 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), pp. 900-905, 2022.

[11] S. Zhang, X. Li, C. Zhang, "Neural Network Quantization Methods For Voice Wake Up Network," Journal of Physics: Conference Series Vol. 1871, No. 1, pp. 012049, 2021.

[12] https://github.com/apache/tvm-rfcs/blob/main/rfcs/0006-AMP_pass.md

[13] M. H. Shin, I. K. Ye, D. W. Lee, "Performance Analysis on TVM Optimization for AI Framework in Autonomous Vehicles," Institute of Embedded Engineering of Korea (IEMEK), 2021 (in Korean).

[14] L. Zheng, C. Jia, M. Sun, Z. Wu, C. H. Yu, A. Haj-Ali, Y. Wang, J. Yang, D. Zhuo, K. Sen, "Ansor: Generating High-performance Tensor Programs for Deep Learning," Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation, pp. 863-879, 2020.

[15] M. H. Shin, I. K. Ye, D. W. Lee, "A Study on the Effect of Low-level Code Optimization for DNNs via TVM Optimization Performance Analysis," Korea Institute of Military Science and Technology (KIMST), 2021 (in Korean).

[16] M. H. Shin, L. K. Ye, D. W. Lee, "A Study on TVM for the Embedded Software in Weapon Systems," The Korea Institute of Intelligent Transport Systems Vol. 2022, No. 6, pp. 246-251, 2022 (in Korean).

[17] H. A. Abdelhafez, H. Halawa, K. Pattabiraman, M. Ripeanu, "Snowflakes at the Edge: A Study of Variability Among NVIDIA Jetson AGX Xavier Boards," Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking, pp. 1-6, 2021.

[18] K. He, X. Zhang, S. Ren, J. Sun, "Deep Residual Learning for Image Recognition," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770-778, 2016.

[19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L. C. Chen, "Mobilenetv2: Inverted Residuals and Linear Bottlenecks," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510-4520, 2018.

[20] M. Tan, Q. Le, "Efficientnetv2: Smaller Models and Faster Training," International Conference on Machine Learning, pp. 10096-10106, 2021.

[21] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, L. Fei-Fei, "Imagenet: A Large-scale Hierarchical Image Database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248-255, 2009.

[22] Y. Wei-Wei, J. ZHANG, "Real-Time Drivers' Violation Behaviors Detection Based on Improved YOLOv3-tiny Algorithm-Based on Model Pruning and Half-Precision Acceleration [J]," Computer Systems & Applications Vol. 29, No. 04, pp. 41-47, 2020.

[23] D. Lin, S. Talathi, S. Annapureddy, "Fixed Point Quantization of Deep Convolutional Networks," International Conference on Machine Learning, pp. 2849-2858, 2016.

[24] P. Nayak, D. Zhang, S. Chai, "Bit Efficient Quantization for Deep Neural Networks," 2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS), pp. 52-56, 2019.

[25] S. Yang, R. Wu, M. Wang, L. Jiao, "Evolutionary Clustering Based Vector Quantization and SPIHT Coding for Image Compression," Pattern Recognition Letters Vol. 31, No. 13, pp. 1773-1780, 2010.

[26] R. C. O. Rocha, V. Porpodas, P. Petoumenos, L. F. Góes, Z. Wang, M. Cole, H. Leather, "Vectorization-aware Loop Unrolling with Seed Forwarding," Proceedings of the 29th International Conference on Compiler Construction, pp. 1-13, 2020.

[27] K. Hammond, S. P. Jones, "Profiling Scheduling Strategies on the GRIP Parallel Reducer," Submitted to Journal of Parallel and Distributed Computing, 1991.

[28] S. D. Hammond, C. T. Vaughan, D. Dinge, P. Lin, C. Hughes, C. R. Trott, J. Cook, R. J. Hoekstra, "Sandia ATDM Performance Execution Tools & Analysis," 2018.
[29] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, "Tensorflow: Large-scale Machine Learning on Heterogeneous Distributed Systems," arXiv preprint arXiv:1603.04467, 2016.
[30] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, "Pytorch: An Imperative Style, High-performance Deep Learning Library," Advances in Neural Information Processing Systems Vol. 32, 2019.
[31] J. Bai, F. Lu, K. Zhang, "Onnx: Open Neural Network Exchange," GitHub Repository, pp. 54, 2019.
[32] https://gist.github.com/masahi/e4c611694e3dfd307a8b6bba45eb1658
[33] G. Bradski, "The OpenCV Library," Dr. Dobb's Journal: Software Tools for the Professional Programmer Vol. 25, No. 11, pp. 120-123, 2000.
[34] S32V Vision and Sensor Fusion Evaluation Board. https://www.nxp.com/products/processors-and-microcontrollers/armbased-processors-and-mcus/s32-automotive-platform/s32v-vision-andsensor-fusion-evaluation-board:SBC-S32V234
[35] D. H. Son, H. Y. Lee, D. H. Im, "Development of High Reliable Real-Time Operating System (RTWORKS) Based on Partitioning and Application of Weapon System," Communications of the Korean Institute of Information Scientists and Engineers Vol. 34, No. 10, pp. 53-59, 2016.

Cheonghwan Hur (허 청 환)

2017 Computer Engineering from Inha University (B.S.)
2019 Electriccal and Computer Engineering from InhaUniversity (M.S.)
2023 Electriccal and Computer Engineering from InhaUniversity (Ph.D.)
2023~RTST (Senior Researcher)
Field of Interests: Deep Learning, Machine Learning, Object Detection
Email: princehur@rtst.co.kr

Ikhee Shin (신 익 희)

2016 Computer Engineering from Chungnam National University (B.S.)
2018 Computer Engineering from Chungnam National University (M.S.)
2021 Computer Engineering from University of Science and Technology (Ph.D. Candidate)
2021~RTST (Senior Researcher)
Field of Interests: Deep Learning, Machine Learning, Object Detection
Email: ihshin@rtst.co.kr

Minhae Ye (예 민 해)

2018~RTST (Senior Researcher)
Field of Interests: Deep Learning Model Architecture
Email: wiz.sensitive@rtst.co.kr

Daewoo Lee (이 대 우)

2002 Electrical Engineering from KAIST (B.S.)
2014 Compute Science from KAIST (Ph.D.)
2018~RTST (AI Team Lead)
Career:
2014~2018 Senior Researcher, NSR
Field of Interests: Deep Learning, Machine Learning, Object Detection
Email: dwlee@rtst.co.kr