ARM 기반 IoT 장치에서 효율적인 딥 러닝 수행을 위한 BLAS 및 신경망 라이브러리의 성능 및 에너지 비교

이하윤°, 신동군
성균관대학교 전자전기컴퓨터공학과
lhy920806@skku.edu, dongkun@skku.edu

Performance and Energy Comparison of Different BLAS and Neural Network Libraries for Efficient Deep Learning Inference on ARM-based IoT Devices

Hayun Lee°, Dongkun Shin
Department of Electrical and Computer Engineering, Sungkyunkwan University

요 약

기존에 IoT 장치에서 딥 러닝을 수행하기 위해 주로 클라우드 컴퓨팅을 사용했다. 그러나 클라우드 컴퓨팅을 사용할 경우 연결을 보장할 수 없고, 통신을 위한 에너지 소모, 그리고 보안에 대한 취약성이 문제가 된다. 이와 같은 문제점을 해결하기 위해 최근 IoT 장치 내에서 딥 러닝을 수행하기 위한 시도가 진행되고 있다. 이 시도들은 주로 IoT 장치를 위한 연산량이 적은 딥 러닝 모델 또는 압축 기법 등을 제안한다. 본 논문에서는 실제로 다양한 하드웨어 구성을 가진 IoT 장치에서 라이브러리 간의 성능을 측정하고, 분석한다. 또한, 적절한 라이브러리를 사용하는 것만으로도 속도와 에너지 효율이 최대 9.43배, 26.78배까지 상승하는 것을 보여준다.

1. 서 론

딥 러닝은 이미지 분류, 자연어 처리, 자율주행차 등 다양한 분야에서 우수한 성능을 보인다. 최근 연구에서 고성능 GPU 또는 가속기에서 딥 러닝 수행을 가속하기 위해 많은 알고리즘 및 하드웨어 구조가 제안되었다. 그러나 기존 IoT 장치를 가속하기 위한 연구는 상대적으로 적다.

이전에는 IoT 장치에서 딥 러닝 응용을 수행하기 위해 주로 클라우드 컴퓨팅을 사용했다. 그러나 클라우드 컴퓨팅을 사용하게 되면 세 가지 문제점이 존재한다. 첫 번째, 불안정한 통신환경과 예측 불가능한 응답 시간으로 인해 서비스 제공에 차질이 발생할 수 있다. 두 번째, 통신을 위해 필요한 에너지가 IoT 장치만으로 수행할 때보다 클 수 있다[1]. 마지막으로, 클라우드로 데이터를 올려야 하므로 보안에 취약하다. 이와 같은 문제점들을 해결하기 위해 IoT 장치에서 딥 러닝을 수행할 필요가 있다.

IoT 장치는 제한된 전력과 낮은 계산 성능을 가지고 있다. 이전 연구에서는 IoT 장치에서 딥 러닝을 수행하기 위하여 최적화된 네트워크 모델[2,3]과 네트워크 압축 기법 등이 제안되었다. 이들은 모두 딥 러닝 수행을 위한 모델의 크기와 곱셈 연산의 수를 줄이는 것에 초점을 맞춰 연구되었다.

그러나, 같은 구조의 네트워크와 압축 기법을 사용하더라도, BLAS 및 신경망 라이브러리에서 실제로 어떤 명령어를 수행하는지에 따라 수행 시간과 에너지가 달라진다. 기존 연구에서는 여러 라이브러리의 수행 시간, 에너지를 분석하지 않았다. 본 논문에서는 다양한 IoT 장치, 딥 러닝 모델, 라이브러리에 대해 수행 시간 및 에너지를 측정한다. 측정에 사용되는 장치들은 표 1과 같이 다양한 하드웨어 구성 및 특성을 갖는다. 이후 장치-라이브러리별 성능 분석을 통해 기존 대비 속도, 에너지 효율을 비교한다.

표 1 다양한 하드웨어 구성 및 지원을 가진 IoT 장치들

	Raspberry Pi 3 *(RPI3)*	ODROID XU3 *(XU3)*	Jetson TX2 *(TX2)*
CPU	Cortex-A53 (quad core)	Cortex-A15 (quad core)	Cortex-A57 (quad core)
		Cortex-A7 (quad core)	Denver2 (dual core)
GPU		Mali T-628 MP6	256-core Pascal
NEON	✓	✓	✓
OpenCL		✓	
CUDA			✓

2. 관련 연구

최근 딥 러닝은 주로 CNN(Convolutional Neural Network)을 수행한다. 그러나 이전에는 CNN 모델이 크기가 컸기 때문에 IoT 장치에서 사용하기에 적합하지 않았다. 이를 해결하기 위해 SqueezeNet[2], MobileNet[3]과 같은 경량 CNN 모델이 제안되었다. 이들은 각각 fire 모듈과 depthwise separable 컨볼루션을 통해 연산량과 모델의 크기를 획기적으로 줄인다.

딥 러닝 모델을 수행하기 위해서 딥 러닝 프레임워크들은

"본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW 컴퓨팅산업원천기술개발사업(SW스타랩)의 연구결과로 수행되었음" (IITP-2017-0-00914)

하드웨어 가속을 위해 BLAS (Basic Linear Algebra Subprograms) 또는 신경망 라이브러리를 사용한다. 이전부터 BLAS 라이브러리를 비교하는 연구는 존재했다[4,5]. 이들은 모두 BLAS 라이브러리 자체의 성능을 비교하지만, 실제 딥 러닝 모델에 적용했을 때에 대해 고려는 없다.

다양한 장치에서 딥 러닝 모델의 수행을 비교하는 연구도 있었다[6]. 이 연구에서는 다양한 장치에 대해 신경망의 수행 시간과 에너지 소모를 측정하였다. 그러나 실험에서 CPU, GPU, DSP가 사용하는 라이브러리가 고정되어 있어 다양한 라이브러리 간의 성능을 비교하지는 않았다.

본 논문에서는 ARM 기반의 IoT 장치에서 사용될 수 있는 다양한 BLAS 및 신경망 라이브러리에 대하여 모델 수행 단위로 수행 시간과 에너지를 비교하고 결과를 분석한다.

3. BLAS / 신경망 라이브러리

BLAS 라이브러리는 기본적인 벡터와 매트릭스 연산을 수행하기 위해 최적화된 라이브러리이다. 일반적으로 컨볼루션 연산을 직접 수행하는 것은 시간이 오래 걸리기 때문에, 그림 1과 같이 GEMM(GEneral Matrix Multiplication)을 사용하여 컨볼루션을 수행한다. CNN 대부분은 컨볼루션 레이어로 구성되어 있으므로, CNN의 성능을 위해 잘 최적화된 BLAS 라이브러리의 GEMM을 사용하는 것이 중요하다.

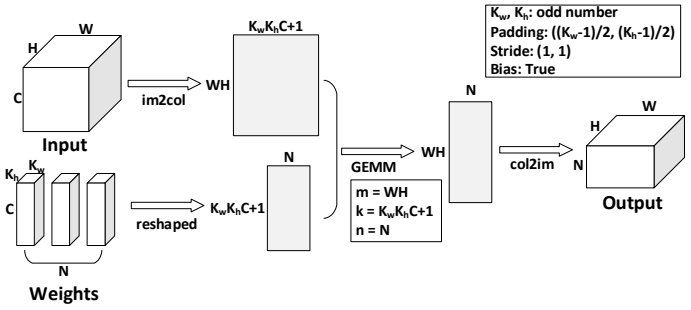

그림 1 GEMM을 사용한 컨볼루션 연산

그림 2는 딥 러닝 모델 수행을 위한 소프트웨어 스택을 보여준다. 대표적인 BLAS 라이브러리에는 OpenBLAS[7], CLBlast[4], cuBLAS가 있다. OpenBLAS는 CPU에서 가속을 수행하며 ARMv8부터 NEON을 지원한다. [1] CLBlast는 OpenCL을 지원하는 GPU에서, cuBLAS는 CUDA를 지원하는 NVIDIA GPU에서 가속을 수행한다.

신경망 라이브러리는 딥 러닝 워크로드에 최적화 된 함수를 제공하는 라이브러리이다. 대표적인 신경망 라이브러리에는 ArmCL(ARM Compute Library)[8], cuDNN이 있다. ArmCL은 ARM CPU와 GPU에서, cuDNN은 CUDA를 지원하는 NVIDIA GPU에서 가속을 수행한다.

또한, 다양한 환경의 장치에서 최적화된 딥 러닝 수행 코드를 생성하는 NNVM/TVM[9] 프레임워크도 제안되었다. 이후 실험에서는 앞서 소개한 라이브러리와 함께 해당 프레임워크를 사용하여 성능을 측정한 후 분석한다.

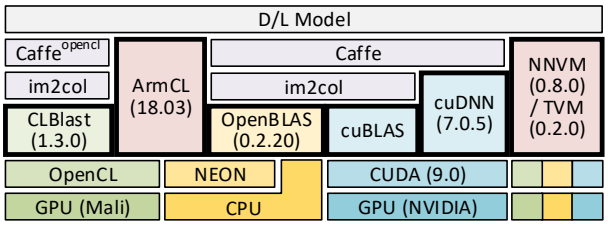

그림 2 딥 러닝 모델 수행을 위한 소프트웨어 스택[2]

4. 실 험

4.1 실험 환경

실험을 위해 표 1에서 언급된 세 가지 장치와 그림 2에 있는 여섯 가지 라이브러리를 사용하였다. 그림 2의 괄호는 실험에 사용된 버전을 의미한다. 각 장치에서 모든 라이브러리를 사용할 수 없으므로, 사용 가능한 라이브러리만을 사용하여 비교 및 분석하였다. 첫 번째 실험에서는 GEMM 성능을 분석하고, 두 번째 실험에서는 딥 러닝 모델을 수행한 후 성능을 분석한다.

딥 러닝 모델 수행 시 ArmCL과 TVM을 제외하고는 딥 러닝 프레임워크로써 Caffe[10]를 사용했다. 전력 측정을 위해 RPI3에서는 Monsoon Power Monitor를 사용하였고, XU3와 TX2에서는 내장된 power monitor를 사용하였다. OpenBLAS의 스레드 개수는 모두 4개로 통일하였고, TX2의 Denver2 CPU는 사용하지 않는다.

4.2 GEMM 성능 비교

그림 3은 매트릭스 크기에 따라 각 장치-라이브러리별 SGEMM(Single precision floating GEMM) 연산 속도를 보여준다.

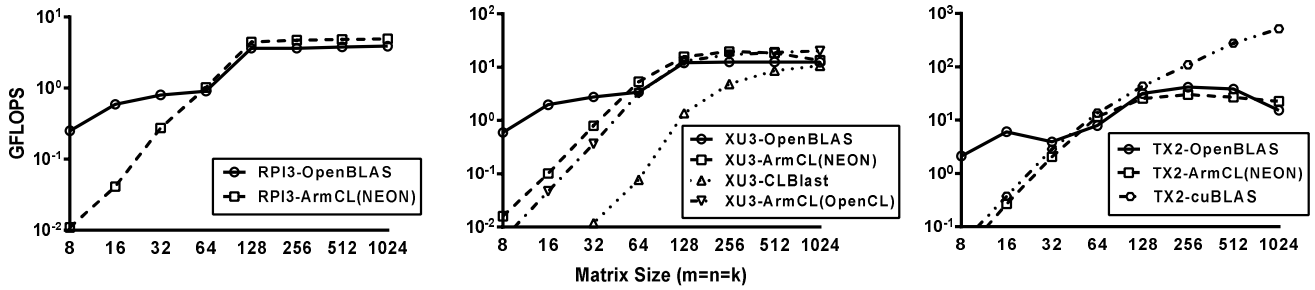

그림 3 장치-라이브러리 별 SGEMM 성능

[1] ARMv7은 성능 문제로 지원하지 않는다.

[2] Caffe: master branch (commit 8645207), Caffe^opencl: opencl branch (commit 74312cf)

모든 라이브러리는 매트릭스의 크기가 커짐에 따라 연산 속도도 같이 증가한다. 특히, OpenBLAS는 매트리스 사이즈가 작을 때에도 다른 라이브러리와 비교하여 높은 연산 속도를 보여준다. 또한, ArmCL(OpenCL)과 cuBLAS를 제외한 나머지 라이브러리들의 SGEMM은 매트리스 크기가 커질수록 SGEMM의 성능이 수렴하게 된다.

4.3 CNN 수행 성능 비교

그림 4와 5는 각각 장치-라이브러리별 CNN을 수행하여 전력 소모, 그리고 수행 시간 및 에너지 효율을 보여준다. 실험에서 사용된 CNN 모델은 SqueezeNet(Sq) v1.1과 MobileNet(Mo) v1이다. 그림 3의 Big/Little은 각각 XU3의 A15/A7의 전력 소모를 나타낸다. 또한, TVM은 RPI3에서 CPU를, XU3와 TX2에서는 GPU를 가속한다.

그림 4에서 XU3의 전력 소모는 CPU를 사용할 때보다 GPU를 사용할 때 현저히 적다. 그러므로 그림 5에서 XU3-CLBlast가 XU3-ArmCL(NEON)보다 속도는 느리지만, 에너지 소모는 적은 현상이 나타난다. TX2는 XU3와는 다르게 GPU의 전력 소모가 큰 편이다.

그림 4에서 TX2-TVM과 다른 라이브러리와 비교하여 GPU의 전력 소모가 높은 편이다. TX2-TVM이 Sq에서 TX2-cuDNN보다 더 낮은 에너지 효율을 보이는데, 이는 TVM의 높은 GPU 활용도로 인해 성능은 좋지만 GPU 활용도만큼 연산을 효율적으로 수행하지 않았음을 의미한다. 또한, TX2-TVM이 Sq와 다르게 Mo에서 독보적인 성능을 보이는데, 그 이유는 현재 Caffe에 depthwise separable 컨볼루션을 위한 최적화가 되어 있지 않기 때문이다.

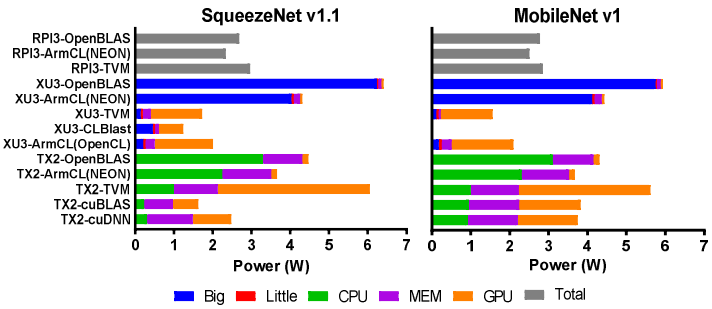

그림 4 장치-라이브러리별 CNN 모델 수행에 따른 전력 소모[3]

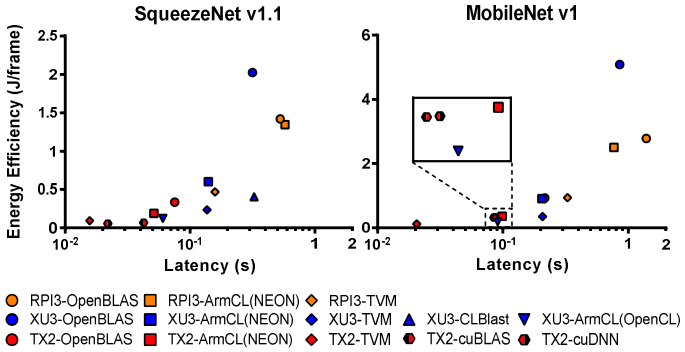

그림 5 장치-라이브러리 별 CNN 모델 속도 및 에너지 효율

표 2는 그림 5의 결과를 바탕으로 속도와 에너지 측면에서의 향상 결과를 보여준다. 베이스라인은 모두 Caffe 프레임워크에서 OpenBLAS 또는 cuDNN을 사용한 것으로 가정하였다. 이 때, 앞선 실험의 결과를 바탕으로 각 장치마다 속도 향상이 가장 높은 라이브러리를 선택하였고, TX2에서 Sq를 수행할 때를 제외하고는 속도와 에너지 효율이 모두 상승하였다.

표 2 최적화 된 라이브러리를 통한 성능 향상

Device	RPI3		XU3		TX2	
Model	Sq	Mo	Sq	Mo	Sq	Mo
Baseline	Caffe + OpenBLAS / cuDNN					
Optimized	TVM		ArmCL(OpenCL)		TVM	
Speed up	3.33x	4.24x	5.21x	9.43x	1.40x	4.26x
Energy	3.02x	4.13x	16.64x	26.78x	0.57x	2.84x

5. 결 론

본 논문은 딥 러닝을 수행하기 위한 다양한 라이브러리를 서로 다른 하드웨어 조건을 갖춘 IoT 장치에서 속도와 에너지를 비교 및 분석한다. 또한, 실험 결과를 통해 자신의 하드웨어에 맞는 최적의 라이브러리를 사용하는 것만으로도 속도와 에너지 효율이 최대 9.43배, 26.78배까지 상승하는 것을 보여준다.

참고문헌

[1] Y. Kang, et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ACM, 2017.

[2] F. N. Iandola, et al., "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size," arXiv preprint arXiv:1602.07360, 2016.

[3] A. G. Howard, et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv:1704.04861, 2017.

[4] C. Nugteren, et al., "CLBLast: A tuned openCL BLAS library," arXiv preprint arXiv:1705.05249, 2017.

[5] F. Li, et al., "CPU versus GPU: which can perform matrix computation faster—performance comparison for basic linear algebra subprograms," Neural Computing and Applications, pp. 1-13, 2018.

[6] N. D. Lane, et al., "An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices," Proceedings of the 2015 International Workshop on Internet of Things towards Applications, ACM, 2015.

[7] https://github.com/xianyi/OpenBLAS

[8] https://github.com/ARM-software/ComputeLibrary

[9] T. Chen, et al., "TVM: End-to-End Optimization Stack for Deep Learning," arXiv preprint arXiv:1802.04799, 2018.

[10] https://github.com/BVLC/caffe

[3] MobileNet v1.1의 XU3-CLBlast 실험은 메모리 부족으로 실행 불가능 하여 제외하였다.