Docker 기반 이기종 엣지 환경에서의 모델 추론 성능 측정 프로그램 구현 및 평가

김성우 [1*], 김은지 [2*], 이종률 [3], 문용혁 [3,4**]
[1] 인하대학교 전자공학과
[2] 인천대학교 정보통신공학과
[3] 한국전자통신연구원 인공지능연구소
[4] 과학기술연합대학원대학교 한국전자통신연구원스쿨
tjddn0402@inha.edu, dmswldpvm@inu.ac.kr, {jongryul.lee, yhmoon}@etri.re.kr

A Docker-based Evaluation Program for Model Inference Performance on Heterogeneous Edge Environments

Seong-Woo Kim[1*], Eun-ji Kim[2*], Jong-Ryul Lee[3], Yong-Hyuk Moon[3,4**]
[1]Dept. of Electronic Engineering, Inha University
[2]Dept. of Information and Telecommunication Engineering, Incheon National University
[3]Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea
[4]University of Science and Technology (UST), Daejeon, Korea

요 약

최근 딥러닝 기술이 모바일 기기에 활발히 적용됨에 따라 다양한 엣지 디바이스에서 신경망 모델의 추론 성능을 측정하는 것이 중요해지고 있다. 하지만 디바이스 별 환경 구성과 런타임별 모델 변환 방식이 다르기 때문에 이를 실제로 수행하는 것은 많은 시간을 필요로 한다. 따라서 본 논문에서는 이기종 환경을 고려하여 추론 성능을 측정할 수 있는 Docker 기반의 프로그램을 구현하였고, 이를 이용하여 다양한 엣지 디바이스에서 최신 모델들의 추론 성능을 측정하였다. 또한, 본 프로그램으로 확보 가능한 추론시간 데이터 기반 추론 성능 예측 연구의 사전 연구로서, 대표적 경량 모델인 MobilenetV1 에 대한 연산자별 프로파일링을 수행하여 추론시간의 변화 양상을 관찰하였다.

1. 서론

소형 스마트 기기 및 딥러닝 기술이 등장한 이후 다양한 엣지 환경에서의 모델 추론을 위한 연구 및 개발이 활발하게 이루어지고 있다. 딥러닝 모델 개발은 모델을 학습시키는 과정과 모델을 디바이스에 배치하여 추론을 수행하는 과정으로 나눌 수 있다[1]. 추론 수행과정에서 활용되는 엣지 디바이스의 경우 전력 및 연산 등의 성능 제약을 가지고 있어서 제한된 환경에서도 잘 동작할 수 있도록 모델을 경량화하는 연구가 활발히 진행되고 있다. 또한, 같은 맥락에서 효과적인 모델의 선정을 위해 실제 디바이스 상의 모델의 추론 성능을 정확하게 측정하기 위한 연구도 등장하고 있다[2], [3]. 그러나 이러한 방향성에도 불구하고, 여전히 모델의 추론 성능 측정은 엣지 디바이스에 런타임 환경 구축 및 모델 변환 등의 복잡한 작업 요구로 수행하기 굉장히 번거롭고, 이로 인해 관련 연구 및 개발 진입에 있어서 다소 높은 장벽을 만들어내고 있다.

본 연구에서는 Docker 를 기반으로 이기종 엣지 환경을 지원하는 신경망 모델의 추론 성능 평가 프로그램을 구현하고, 이를 이용하여 다양한 런타임과 엣지 디바이스에서 최신 이미지 분류 모델들에 대해 벤치마킹을 실시하였다. 또한, 구현된 프로그램을 통해 광범위하게 모델의 추론시간 데이터를 확보할 수 있는데, 이를 활용하면 임의의 신경망 모델에 대해 타겟 디바이스 및 런타임 환경에서의 추론시간을 예측하는 모델을 구축할 수 있다. 이를 위한 사전 연구로서 런타임에서 직접 지원되는 프로파일링을 통해 각 런타임에서 발생하는 연산자 fusion 을 관찰하고, 연산자별 추론시간을 측정하였다. 그 결과, 동일한 모델

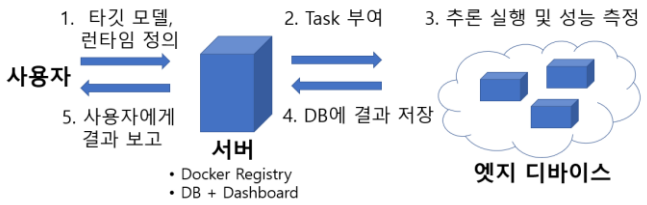
* 공동 주저자
** 교신저자

(MobileNetV1) 내부 연산자에 대해 디바이스 및 런타임별로 유의미한 추론시간 변화를 관찰하여 추론 성능 예측 시 이를 고려할 필요성을 확인하였다.

2. Docker 기반 신경망 모델 벤치마킹 프레임워크

Docker 는 host OS 상에서 동작하는 컨테이너 기반의 가상화 도구이다. 본 연구에서 Docker 를 기반으로 한 평가 프로그램을 구축한 이유는 타겟 런타임 및 디바이스의 아키텍처에 대한 추론 환경을 구축하여 이미지를 레지스트리에 저장하면, 같은 하드웨어 플랫폼을 공유하는 이기종 디바이스에 런타임 환경을 따로 구축하지 않고도 추론 성능을 측정할 수 있기 때문이다. 또한 필요한 경우, 사용자가 원하는 환경으로 미리 Docker 컨테이너를 구성하고 레지스트리에 추가적으로 등록하여 사용할 수 있다. 이러한 특성은 향후 사용자에 의한 지원 디바이스 및 런타임의 확장을 용이하게 한다.

2.1. 시스템 구조



(그림 1) 신경망 모델 추론 성능 평가 프로그램 동작 과정.

본 프로그램은 서버, 에이전트, 사용자로 구성되며 동작 과정은 아래와 같다. 에이전트는 엣지 디바이스 상에서 구동되며, 엣지 디바이스에서 평가 프로그램을 호출하는 역할을 수행하고, 사용자는 라이브러리를 통해 서버에 모델에 대한 평가를 요청한다.

- 프로그램 동작 과정

1) 사용자는 이미 학습된 Keras 모델을 준비하고 엣지 디바이스를 지정하여 서버로 평가를 요청한다. 여기서 평가 요청은 하나의 태스크로 정의되고, 태스크 수행을 위한 모델 로드 및 추론시간 평가 프로그램을 포함한다. 2) 서버에서는 요청된 태스크를 사전에 구성된 Docker 이미지와 함께 에이전트에 전달한다. 3) 에이전트는 서버로부터 전달받은 Docker 이미지를 로드 하여 평가 프로그램을 실행한다. 4) 태스크 수행이 끝난 후 에이전트는 결과를 서버에 전달한다. 5) 서버에서 사용자에게 결과를 전송한다.

3. 모델 추론 성능 평가 실험

3.1. 벤치마크 구성

타겟 모델은 알려진 최신 모델들을 이용하였고, 런타임으로는 TensorFlow-Lite, TensorRT 를 이용했다. 벤치마크를 실시한 하드웨어 환경은 표 1 과 같다.

<표 1> 벤치마킹에 사용된 디바이스

	Jetson AGX Xavier	Jetson Nano	Raspberry pi 4 (RPI)
CPU	8-core ARMV8 64bit	Quad-core ARM v8 64bit	Quad-core ARM v8 64bit
GPU	512 cores, Volta	128 cores, Maxwell	-
RAM	32GB	4GB	8GB

3.2. 벤치마크 방법

TensorFlow-Lite 런타임의 경우 모델 변환을 서버에서 수행한 후 디바이스에 모델을 제공하였다. 반면 TensorRT 에서는 디바이스의 GPU 아키텍처와 동일한 환경에서 모델 변환을 수행해야 하는 제한이 있어서 부득이 서버에서는 ONNX 모델로 변환하고, 최종적으로 TensorRT 모델로의 변환은 엣지에서 수행하였다. 표 1 의 벤치마크 결과는 모델 추론 수행시간만 포함한다. 결과값은 30 회 반복 측정 결과의 평균이다.

3.3. 벤치마크 결과

표 2 에서 나타난 결과에서 추론 성능의 간접적 지표로 많이 사용되는 FLOPs 가 실제 추론시간에 그렇게 비례하지 않음을 확인할 수 있었다. 예를 들어, EfficientNetB0 의 경우, MobileNetV1 보다 FLOPs 가 작지만 추론시간을 비교했을 때 평균적으로 2 배 이상 시간이 소요되는 것을 확인할 수 있다. 또한, TensorRT 런타임에서는 ResNet50 의 FLOPs 가 EfficientNetB0 보다 약 10 배 크지만 추론시간은 더 작게 관찰되었다.

한가지 흥미로운 것은 MobileNetV1 과 MobileNetV2 간 성능 양상이 RPI 와 Xavier 간에 정반대로 나타난다는 점이다. 이는 MobileNetV2 와 같이 Residual/Skip Connection 을 보유하여 비교적 높은 메모리 사용률을 요구하는 모델을 RPI 가 처리할 때, 낮은 메인 메모리로 인해 추론속도가 현저히 감소하기 때문이다.

모든 결과에서 알려진 대로 TensorRT 런타임을 사용한 경우가 TensorFlow-Lite 보다 월등히 빠른 추론 속도를 보였다. 이는 TensorFlow-Lite 의 경우 디바이스의 CPU 에서 추론이 일어나지만, TensorRT 런타임은 GPU 의 병렬 연산을 이용하기 때문이다.

4. Operator 별 추론시간 프로파일링

4.1. 프로파일링 방법

효율적인 추론을 위해 최적화를 지원하는 대표적인 런타임 환경인 TensorFlow-Lite 와 TensorRT 에 대하여 operator(연산자)별 프로파일링 실험을 수행하였다. 본 실험에서 모델은 MobileNet1 이 사용되었고, operation

<표 2> 벤치마킹 프레임워크를 사용하여 측정한 추론시간(ms) 및 FLOPs

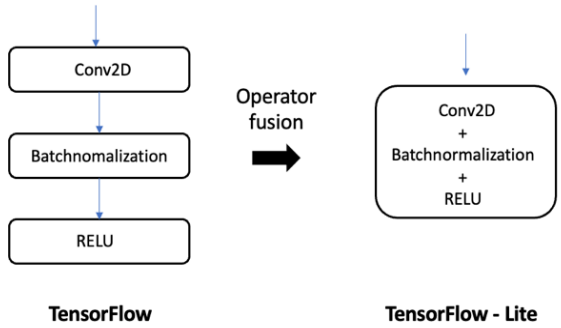
	TensorFlow-Lite			TensorRT		FLOPs
	Raspberry pi 4	Jetson Xavier	Jetson Nano	Jetson Xavier	Jetson Nano	
EfficientNetB0	372.59	147.04	333.19	3.48	12.25	0.783B
MobileNetV1	144.46	114.63	124.67	0.67	4.84	1.137B
MobileNetV2	193.31	68.19	107.88	1.95	4.99	0.602B
ResNet50	423.36	744.49	613.85	1.65	7.68	7.728B
DenseNet201	1,362.82	847.91	796.43	6.37	19.63	8.585B

fusion 을 고려하여 연산자별 추론시간 기여 비율을 계산하였다. 참고로 TensorFlow-Lite 에서 연산자별 추론시간을 측정하기 위해 TensorFlow 에서 제공하는 벤치마크 툴을 Bazel 을 통해 빌드하여 사용하였다. TensorRT 의 경우에는 TensorRT 에서 자체적으로 지원하는 IProfiler 클래스를 이용하여 프로파일링 하였다.

4.2. 프로파일링 결과 및 분석

4.2.1. TensorFlow-Lite

MobileNetV1 모델을 TensorFlow-Lite 버전으로 변환한 후에 프로파일링한 결과, 모델이 TensorFlow-Lite 로 변환될 때 기존 TensorFlow 연산자들이 합쳐지거나 여러 개의 연산자들로 분해되는 공통적인 패턴을 찾아 낼 수 있었다. 아래의 그림 2 는 TensorFlow 의 convolution, batch normalization, relu 연산자가 TensorFlow-Lite 로 변환되며 하나의 Conv2D 로 fusion 되는 예시를 보여준다.



(그림 2) TensorFlow 에서 TensorFlow-Lite 로 모델 변환 시 3 개의 연산자가 Fusion 되어 하나의 연산이 됨.

<표 3> TensorFlow-Lite MobilenetV1 fusion operator

TensorFlow	TensorFlow-Lite
Conv2D+BN+Relu	Conv2D
DWConv2D+BN+Relu	DWConv2D
GAP2D	Mean + Shape+StrideSlide+Pack
Conv2D	Conv2D+Shape+StrideSlice+Pack

표 3 은 TensorFlow-Lite 런타임을 위해 모델 변환 시 MobileNetV1 에 대한 fusion operator 들을 정리한 것이다. 여기서 Conv2D 는 2D convolution, BN 은 batch normalization, DWConv2D 는 depth-wise convolution, GAP2D 는 global average pooling 을 의미한다.

각각의 디바이스에서 TensorFlow-Lite Benchmark tool 을 이용하여 MobilenetV1 모델에 대해 프로파일링 한 결과, 가장 오랜 시간을 소요하는 상위 연산자 4 개가 전체 추론시간의 99% 이상을 차지하는 것을 확인하였다. 아래의 표 4 는 이러한 연산자 top-4 개에 대해 전체 추론시간 대비 차지하는 비율을 디바이스 별로 정리한 것이다. 본 결과에서는 동일 모델, 동일 런타임을 이용하여 추론을 했음에도 불구하고 디바이스에 따라 연산자별 소요되는 시간 비율이 다름을 확인할 수 있다. RPI 의 경우 DWConv2D 연산자에서 다른 디바이스보다 전체 추론시간 대비 더 많은 시간을 소요함을 알 수 있고, Mean 연산자에서는 더 낮게 소요함을 확인할 수 있다.

<표 4> TensorFlow-Lite 에서 MobilenetV1 프로파일링 결과 전체 추론시간 대비 비중 Top-4 연산자 (단위: %)

TensorFlow-Lite	RPI	Nano	Xavier	실행횟수
Conv2D	64.99	70.01	77.4	15
DWConv2D	31.14	22.21	16.81	13
Pad	2.49	3.54	3.20	4
Mean	1.32	4.16	2.37	1
합계	99.94	99.92	99.57	-

4.2.2. TensorRT

아래의 표 5 는 Jetson AGX Xavier 와 Jetson Nano 에서 TensorRT 런타임을 통한 모델 최적화를 수행했을 때, MobileNetV1 에 대한 fusion operator 을 관찰한 것이다. 표 5 에서 자주 등장하는 Depthwise 는 depth-wise convolution 을 의미한다.

<표 5> MobileNetV1 을 TensorRT 로 변환할 때 발생하는 fusion operator 및 변화

TensorFlow	ONNX	TensorRT
DWConv2D BN ReLU	Depthwise FusedBatchNorm Relu6	Depthwise+Relu6
ZeroPadding2D DWConv2D BN ReLU	Pad Depthwise FusedBatchNorm Relu6	Pad+Depthwise+Relu6
Conv2D BN ReLU	Conv2D FusedBatchNorm Relu6	Conv2D + Relu6
GAP2D	Mean Mean_Squeeze	Mean

TensorRT 런타임의 경우 Keras 모델은 ONNX 모델을 거쳐서 TensorRT 모델로 변환된다. 따라서 Keras 모델에서 ONNX 모델로 변환될 때와 ONNX 모델이

TensorRT 모델로 변환될 때 각각 어떤 변화가 일어나는지 각각의 경우를 관찰했다. 표 5 에 등장하는 세 종류의 convolution 연산에 대해 모두 Keras 모델의 Batch Normalization operation 이 ONNX 모델에서 FusedBatchNorm operation 으로 바뀐 후 TensorRT 로 변환되며 사라지는 현상이 관찰되었다. 이는 TensorRT 에서 batch normalization 연산을 명시적으로 지원하지 않아 대신에 IElementWiseLayer 클래스를 통해 직접 구현하여 사용해야 하기 때문이다. 또한, TensorFlow 의 ReLU 연산이 ONNX 에서 ReLU6 로 clipping 되어 바뀐 뒤 TensorRT 런타임에서 Conv2D, Depthwise 연산자와 fusion 되었다.

TensorFlow 에서 ONNX 로 모델이 변환될 때 Transpose 연산은 새로 생기는 반면 Dropout 연산은 사라졌고, 이는 TensorRT 로 변환될 때도 유지되는 것을 확인할 수 있다. 또한 Reshape 연산자는 TensorRT 에서 지원하지 않아 사라짐을 확인하였다.

표 6 은 두 디바이스에서 가장 많은 GPU time 을 차지하는 연산자를 순서대로 나열하고, 총 추론시간 대비 각 연산자가 차지하는 비중을 나타낸 것이다. 표 6 에서 Pad + depthwise + Relu6 연산자의 경우 4 번밖에 사용되지 않았지만 9 번 사용된 depthwise + Relu6 연산자보다 더 많은 추론시간을 소요한다는 것을 알 수 있다. 또한 1 번 사용된 BiasAdd 연산자의 경우 Jetson Nano 에서는 전체 추론시간의 4.78%를, Jetson AGX Xavier 에서는 1.50%를 차지했다.

<표 6> TensorRT 에서 MobileNetV1 프로파일링 결과 전체 추론시간 대비 비중 Top-4 연산자 (단위: %)

TensorRT	Nano	Xavier	실행 횟수
Conv2D + Relu6	62.54	71.81	14
Pad + depthwise + Relu6	20.76	14.06	4
depthwise + Relu6	8.90	11.13	9
BiasAdd	4.78	1.50	1
합계	96.98	98.49	-

표에는 나타나지 않았지만, TensorFlow-Lite 런타임과 비교했을 때 Mean 연산자는 Jetson Xavier 와 Jetson Nano 에서 총 추론시간 대비 각각 0.43%, 0.55%의 매우 적은 GPU time 을 차지했다. 이를 통해 런타임별로 각 operation 이 추론시간에 영향을 미치는 정도가 다름을 확인했다. 또한, 동일한 런타임 및 모델이라고 하더라도 기기에 따라 연산자별 추론시간 비율이 다소 달라질 수 있음도 확인할 수 있었다.

종합적으로 위의 결과는 모델의 추론 성능 예측에 있어서 fusion 되는 연산자들의 양상을 살펴볼 필요가 있고, 기기의 종류와 런타임, 그리고 연산자를 함께 고려해야함을 시사한다.

5. 결론 및 향후 연구 계획

본 연구에서는 Docker 기반 이기종 엣지 환경을 지원하는 모델 추론 성능 평가 프로그램을 구축하고, 대표적인 convolutional 모델에 대하여 벤치마킹 실험을 수행하였다. 또한 많이 알려진 런타임인 TensorFlow-Lite 와 TensorRT 에서 경량 모델인 MobileNetV1 에 대해 모델 최적화시 발생하는 fusion operator 패턴을 확인하였고, 프로파일링을 통해 연산자별 추론시간 비율을 측정하였다. 이를 통해 동일 모델에서 런타임과 연산자, 기기에 따라 추론시간 비율의 다양한 양상을 확인할 수 있었다.

향후 연구 계획으로, 본 프로그램을 기반으로 다양한 신경망 모델과 런타임 환경에서 추론 성능에 대한 양질의 학습 데이터를 축적하여 임의의 디바이스와 런타임에서 타겟 모델에 대한 추론시간을 예측할 수 있는 모델을 구성하려고 한다. 이를 실현시키기 위해 fusion operator 을 고려한 연산자별 추론시간 예측 모델을 구축하는[1], [4], [5] 선행연구의 방법론을 도입 및 개선할 계획이다.

Acknowledgement

이 논문은 2022 년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No. 2021-0-00907, 능동적 즉시 대응 및 빠른 학습이 가능한 적응형 경량 엣지 연동분석 기술개발).

참고문헌

[1] Li, En, Zhi Zhou, and Xu Chen. "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy." In Proceedings of the 2018 Workshop on Mobile Edge Communications, 2018, pp. 31-36.

[2] Baller, Stephan Patrick, Anshul Jindal, Mohak Chadha, and Michael Gerndt. "DeepEdgeBench: Benchmarking Deep Neural Networks on Edge Devices." In 2021 IEEE International Conference on Cloud Engineering (IC2E), 2021, pp. 20-30.

[3] Lee, Changsik and Hong, Seungwoo and Hong, Sungback and Kim, Taeyeon. "Performance analysis of local exit for distributed deep neural networks over cloud and edge computing." ETRI Journal, Vol. 42, pp. 658-668, 2020.

[4] Zhang, Li Lyna, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. "Nn-Meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices." In Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services, 2021, pp. 81-93.

[5] Tang, Xiaohu and Han, Shihao and Zhang, Li Lyna and Cao, Ting and Liu, Yunxin. "To Bridge Neural Network Design and Real-World Performance: A Behaviour Study for Neural Networks." In Proceedings of Machine Learning and Systems, 2021, pp. 21-37.