가지치기-양자화 결합을 통한 온디바이스 경량화 실증 연구

최상아*, 권소현○, 이주홍○, 박형곤○

이화여자대학교

컴퓨터공학과*, 전자전기공학과○

{gloody, 2376019, joohong.rheey, hyunggon.park}@ewha.ac.kr

An Empirical Study of On-Device Model Compression via Pruning and Quantization

Shanga Choi*, Sohyun Kwon○, Joohong Rheey○, Hyunggon Park○

Ewha Womans University

Dept. of Computer Science and Engineering*, Dept. of Electronic and Electrical Engineering○

요 약

최근 IoT의 확산으로 실시간 데이터 처리가 가능한 온디바이스 AI의 필요성이 커지고 있다. 본 논문의 선행 연구의 샤플리 값 기반 구조적 가지치기를 실제 임베디드 환경으로 확장하고, 양자화와의 결합 효과를 TensorFlow Lite 및 라즈베리파이 환경에서 실증한다. 실험 결과, 동일 가지치기 수준에서 샤플리 기반 가지치기 방법에 양자화를 결합하였을 때 높은 정확도를 보였다. 이는 샤플리 값 기반 가지치기 후 양자화를 적용하는 파이프라인은 엣지 CPU 환경에서 정확도 유지와 자원 절감을 동시에 달성할 수 있는 실용적 대안임을 보여준다.

Ⅰ. 서론

IoT 기술의 발전으로 네트워크에 연결된 기기의 수가 급증하였다. 이에 따라 전송 지연과 네트워크 접속 장애 등을 완화하기 위하여, 디바이스 자체에서 정보를 처리하는 온디바이스 AI 기술의 필요성이 대두되었다[1]. 가용 자원이 제한적인 엣지 디바이스에서도 딥러닝 모델을 원활하게 운영하기 위해서는 낮은 지연, 적은 연산량과 메모리 사용량이 요구된다[2].

온디바이스 AI를 위한 대표적 기법에는 가지치기(Pruning), 양자화(Quantization), 지식 증류(Knowledge distillation) 등이 있다. 가지치기와 양자화는 온디바이스 제약에서 상보적인 역할을 한다[3]. 가지치기는 모델을 구조적으로 축소하여 파라미터 수와 연산량을 줄이고, 양자화는 남은 가중치의 비트폭을 축소하여 모델 크기와 메모리 대역폭 요구량을 줄이는 데 기여한다.

본 논문은 선행 연구[4]에서 제안한 샤플리 값 기반 가지치기를 실용 환경으로 확장한 실증적 연구이다. 가지치기된 TensorFlow 모델에 정밀도가 다른 두 양자화 기법을 적용하며 TensorFlow Lite(TFLite) 모델로 변환하였다. 변환된 모델을 라즈베리파이에 탑재하여 CPU 단독으로 실행하였다. 이를 통해 가지치기와 양자화의 효용을 평가하고, 엣지 환경에서의 실행 가능성을 확인하였다.

Ⅱ. 본론

1) 중요도 기반 가지치기

샤플리 값은 협력 게임 이론에서 참여자들의 성과를 정량적으로 평가하기 위해 사용하는 값이다. 본 논문에서 사용한 중요도 기반 가지치기 방법[4]에서는 딥러닝 모델의 링크(연결 가중치)들을 추론을 위한 협력 게임의 참여자로 간주하고, 최종 출력 값 추론에 기여한 정도를 샤플리 값으로 정량화한다. 학습이 끝난 딥러닝 모델에 대해 각 링크의 샤플리 값을 계산한 다음, 샤플리 값이 작은 링크부터 가지치기하여 출력 생성에 주요한 역할을 한 링크만 남긴다.

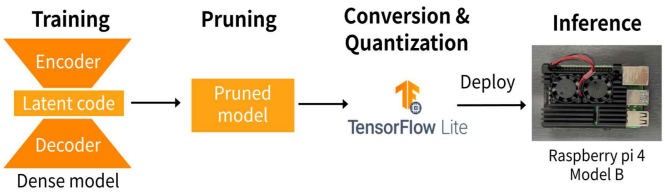

그림 1. 전체 워크플로우

2) 훈련 후 양자화

본 연구에서는 두 종류의 훈련 후 양자화(Quantization After Training; QAT) 기법을 사용한다. 첫 번째 방법은 32bit으로 저장된 가중치를 IEEE float16 (FP16) 포맷[5]으로 양자화하여 floating point 모델로 저장하는 것이다. 이 방법에서는 FP16 가중치를 첫 번째 추론 이전에 float32 (FP32)로 업 샘플링하여 모델 크기를 줄이면서도 정밀도 손실을 최소화한다.

두 번째 방법은 32비트 모델을 동적 범위 양자화(Dynamic range quantization)하여 활성화(Activation) 값을 8비트 정수 (INT8)로 연산하는 것이다. 정수형 수치 연산 이후 부동소수점으로 역 양자화하는 과정을 반복하여 지연 시간을 줄이고 용량을 절감한다. FP16 방식과 달리 신경망에서 나온 출력값들의 범위를 구하여 8비트 정수형 값으로 스케일링하는 추가 작업이 필요하다. INT8 스케일링 공식 S는 다음과 같다.

$$S = (r_{\max} - r_{\min}) \div (2^8 - 1) \quad (1)$$

$r_{\max}$는 최댓값, $r_{\min}$은 최솟값을 의미한다.

3) TFLite 변환 및 디바이스 탑재

본 연구에서는 TensorFlow 프레임워크에서 생성한 모델을 TFLite Converter를 사용하여 보다 작고 효율적인 TFLite 모델로 변환한다. 이 과정에서 FP32 모델을 FP16, INT8로 양자화한다. 이를 위하여 Linux kernel 6.12.34 환경에서 python 3.11.2, tflite_runtime 2.14.0 버전을 사용한다. CPU와 메모리 사용량 측정에 쓰인 psutil 라이브러리 버전은 7.0.0이고, 추론에는 라즈베리파이 4의 XNNPACK/NEON delegate를 적용한다. 구체적인 디바이스 스펙은 표 1과 같다.

Ⅲ. 실험

가지치기와 양자화의 효용을 실증적으로 확인하기 위하여 MNIST 데이터셋[5]을 사용한 분류(Classification) 실험을 진행한다. 베이스라인(Baseline) 모델은 경량화 전 FP32 오토인코더 모델이며, 두 가지 모델 경량화 기법(가지치기, 양자화)를 적용한 결과를 비교한다. 가지치기는 샤플리 값 기반 방법(Shapely value-based pruning)[4]과 가중치의 값이 작은 링크부터 가지치기하는 크기 기반 방법(Magnitude-based pruning)[7]을 대조한다. 양자화는 FP32 모델과 FP16, INT8 동적 범위 양자화 모델을 비교한다.

분류 정확도 측정을 위하여, 오토인코더 출력을 Support Vector Machine (SVM)을 입력한다. SVM 학습에는 Radial Basis Funtion (RBF) 커널을 이용하였으며, 학습 데이터와 테스트 데이터는 분리하였다. 모든 결과는 10회 반복 실험의 평균값이다.

표 2는 가지치기 수준과 양자화 방식에 따른 오토인코더의 분류 정확도(Accuracy, 단위: %)를 나타낸다. 표의 퍼센트는 가지치기 후 남은 링크의 비율을 의미한다. 동일한(혹은 유사한) 가지치기 수준에서 샤플리 기반 가지치기 방식이 크기 기반 가지치기 방식보다 높은 정확도를 보였다. 이는 가지치기 정도가 커질수록 큰 차이를 보였다. 95% 가지치기 수준에서 두 방법 간 차이가 양자화 방식에 따라 12.1%p, 12.85%p, 13.01%p (각각 FP32, FP16, INT8)로 나타났다. 샤플리 값 기반의 가지치기 방법의 경우 양자화 적용 전후의 정확도 차이가 대부분 0.2~0.3%p 이내로 양자화 민감도가 낮은 결과가 나타났다. 이는 샤플리 값 기반의 가지치기 방법을 양자화와 결합하면, 양자화 과정에서 큰 정확도 손실 없이 양자화 전의 모델과 유사한 정확도를 유지할 수 있음을 보여준다.

Ⅳ. 결론

본 논문에서는 샤플리 값 기반 가지치기와 훈련 후 양자화를 결합한 모델을 라즈베리 파이에 탑재하여, 온디바이스 추론을 실증하였다. 실험을 통해 샤플리 값 기반의 가지치기 후 양자화를 적용하는 파이프라인은 양자화 전에 비해 정확도 손실이 거의 없다는 사실을 보였다. 이는 크기 기반 가지치기 방식과 양자화를 결합하였을 때보다 유의미하게 높은 성능이며, 온디바이스 환경에서 정확도와 자원 제약 간의 균형을 안정적으로 달성할 수 있다. 향후에는 가지치기와 양자화의 누적 오차를 피하기 위하여 단계적 가지치기와 재학습 후 양자화 보정(Calibration)을 적용하는 것에 관한 연구를 진행할 예정이다.

Model	Raspberry pi 4 Model B
CPU	ARM Cortex-A72 MP4, 1.8GHz Quad-core
RAM	4 GB

표 1. 디바이스 스펙

		FP32	FP16	INT8
Baseline		97.92	98.40	98.48
Shapley value-based pruning	83%	97.85	98.03	98.04
	88%	97.21	97.19	97.13
	92%	95.96	95.98	95.82
	95%	88.41	88.63	88.24
Magnitude -based pruning	82%	94.31	94.44	94.19
	88%	94.60	88.84	88.82
	91%	83.04	83.10	83.21
	95%	76.20	75.78	75.23

표 2. 가지치기 및 양자화 정도에 따른 모델 성능

ACKNOWLEDGMENT

이 논문은 2021년도와 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No. RS-2021-II210739, 분산/협력AI 기반 5G+ 네트워크 데이터 분석 기능 및 제어 기술 개발, No. RS-2024-00-344830, 6G 네트워크 구조/산업융합 표준기술 개발 및 표준화)을 받아 수행된 연구임.

참 고 문 헌

[1] W. Dingzhu, P. Liu, G. Zhu, Y. Shi, J. Xu, Y. C. Elda and S. Cui, “Task-Oriented Sensing, Computation, and Communication Integration for Multi Device Edge AI,” *IEEE Transactions on Wireless Communication*, vol. 23, No. 3, pp. 2486-2502, 2023.

[2] M. Ramadan, A. Mohammed, K. Shin, and A. Mohammad, "Federated Learning and TinyML on IoT Edge Devices: Challenges, Advances, and Future Directions." *ICT Express,* vol. 11, no. 4, pp. 754-768, 2025.

[3] S. Han, H. Mao, and W. J. Dally, “Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding,” *International Conference on Learning Representations*, 2016..

[4] J. Rheey and H. Park, “SV-SAE: Layer-Wise Pruning for Autoencoder Based on Link Contributions,” *IEEE Access*, vol. 13, pp. 75666-75678, 2025.

[5] "IEEE Standard for Floating-Point Arithmetic," in *IEEE Std 754-2008*, pp.1-70, 29, 2008.

[6] Y. LeCun, C.ortes, and C. Burges (2010), MNIST Handwritten Digit Database. [Online] Available: http://yann.lecun.com/exdb/mnist

[7] S. Han, J.Pool, J. Tran and W. Dally, “Learning Both Weights and Connections for Efficient Neural Network,” *Advances in Neural Information Processing Systems 28*, 2015.