Journal of KIIT. Vol. 21, No. 12, pp. 77-85, Dec. 31, 2023. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2023.21.12.77

Mixed Precision 기반 CNN 추론 연산의 구현 및 성능 분석

이종은*[1], 장경빈*[2], 임승호**

Implementation and Performance Analysis of Mixed Precision-based CNN Inference

Jong-Eun Lee*[1], Kyung-Bin Jang*[2], and Seung-Ho Lim**

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (NRF-2021R1F1A1048026). 이 연구는 한국외국어대학교 교내학술연구비의 지원에 의하여 이루어진 것임.

요 약

최근 딥러닝 기술은 고정밀도 모델로 인해서 정확도가 높아지고 있으나, 복잡한 네트워크의 깊이와 파라미터의 수 증가로 인해서 연산에 필요한 자원 소모 및 프로세싱 시간이 증가한다. 이러한 네트워크 복잡도는 IoT 시스템 및 모바일 시스템과 같은 임베디드 시스템에서 딥러닝 연산을 수행하는 오버헤드가 된다. 따라서, 양자화와 같은 경량 시스템에 맞는 딥러닝 모델을 위한 경량화 연구가 많이 이루어져 왔다. 양자화를 적용하면 정확도 손실이 많이 발생하기 때문에 파라미터를 혼합해서 사용하는 Mixed Precision 적용이 대안이 되고 있다. 본 논문에서는 임베디드 시스템에서 Mixed Precision 기반의 CNN 딥러닝 모델의 추론 연산 방식을 구현하였으며, Mixed Precision 적용에 대한 성능 분석을 하였다. 실험 결과, 연산 복잡도가 높은 몇몇 계층에 양자화를 적용하면 정확도의 손실을 줄이면서 추론 시간이 14%~20%가량 줄어듦을 확인할 수 있었다.

Abstract

Recently, deep learning technology has increased accuracy due to high-precision models, but resource consumption and processing time required for deep learning operation increase due to the increase in the number of parameters and the depth of complex networks. Such network complexity becomes an overhead for performing deep learning operations in embedded systems such as IoT systems and mobile systems. A lot of research on lightweighting for deep learning models suitable for systems such as quantization has been made. However, since accuracy loss occurs when quantization is applied, mixed precision application that uses mixed parameters is becoming an alternative. In this paper, we implemented an inference operation method of a CNN deep learning model based on Mixed Precision for embedded system, and analyzed the performance of Mixed Precision-based inference of CNN model. As a result of the experiment, it is identified that the inference time can be much reduced with little loss of accuracy with mixed precision when quantization is applied to only few layers with high computational complexity.

Keywords

deep learning, embedded system, mixed precision, quantization/dequantization

* 한국외국어대학교 컴퓨터공학부 학사과정
- ORCID[1]: http://orcid.org/0009-0000-4431-3170
- ORCID[2]: http://orcid.org/0009-0000-9782-2841
** 한국외국어대학교 컴퓨터공학부 교수(교신저자)
- ORCID: http://orcid.org/0000-0003-3096-0785

· Received: Aug. 22, 2023, Revised: Sep. 19, 2023, Accepted: Sep. 22, 2023
· Corresponding Author: Seung-Ho Lim
Division of Computer Engineering, Hankuk University of Foreign Studies, Korea
Tel.: +82-31-330-4704, Email: lim.seungho@gmail.com

Ⅰ. 서 론

최근, 인공지능(AI, Artificial Intelligence) 기술 및 딥러닝(Deep learning) 알고리즘의 발전과 더불어 이를 실행 가능하게 하는 프로세싱 기술의 발전으로 인해서 많은 분야에서 인공지능 기술 활용이 점점 늘어가고 있다. CNN(Convolutional Neural Network)과 같은 딥러닝 알고리즘은 알고리즘의 고도화로 인해서 인간의 인식률보다 높은 정확도를 보여주고 있다. 그러나 이러한 딥러닝 신경망 모델에서 그 정확도와 성능을 높이기 위해서 딥러닝 모델에 대한 네트워크의 깊이가 높아지고 파라미터의 수가 증가함으로 인해서 프로세싱 자원과 메모리 자원 등 딥러닝 연산에 필요한 하드웨어 자원의 요구가 증가하고 있다[1]. 그런데, IoT 시스템, 모바일 장치, 자율주행 자동차와 같은 임베디드 시스템을 활용하는 다양한 분야에서는 제한된 하드웨어 자원에서 인공지능 추론 연산을 수행하기 위해서 딥러닝 모델 및 파라미터의 효율적인 구성 및 연산이 필요하다[2].

효율적인 딥러닝 모델 연산을 위한 딥러닝 모델의 경량화 기술이 많이 연구되고 있으며, 주요 기술로는 양자화, 가지치기, 지식 증류, 경량 네트워크 설계 등의 기술이 있다[1]. 임베디드 시스템에서는 실시간 추론 연산을 수행하기 위해서 위와 같은 경량화 기술을 이용하여 모델의 네트워크 및 파라미터를 최적화하여 모델의 정확도를 유지하면서 실행시간을 줄여서 사용성을 높인다.

양자화의 경우, 일반적으로 딥러닝 모델의 파라미터가 부동소수점(Floating point)을 기반으로 생성되어 연산의 복잡도가 높은데, 부동소수점을 정수형(Integer)으로 변환하여 파라미터 크기 및 연산의 복잡도를 낮춤으로써 딥러닝 모델을 최적화시키는 방식이다. 양자화는 학습 후 양자화와 학습 중 양자화를 하는 방법이 있으며 fp32(32bit floating point) 데이터를 주로 fp16, int8(8bit Integer) 등으로 변환하면서 파라미터 삭제 없이 모델의 크기를 줄임으로써 딥러닝 모델의 정확도에 대한 성능저하가 적다[3][4]. 하지만 정밀도를 낮추면 학습 및 추론 오차가 발생하며 심한 경우 학습 진행이 되지 않을 수 있다[6]. 모델의 양자화에 따른 정확도 감소의 주요 요인으로는 int8을 이용해 합성곱 연산을 하게 되면 오버플로(Overflow)가 일어나 필연적으로 학습 중 오차가 발생할 수 있고, 이에 따라 학습이 진행되지 않는 문제가 발생할 수 있다.

파라미터 양자화에 따른 오차 발생 및 성능저하가 발생하는 문제는 Mixed Precision으로 완화할 수 있다. Mixed Precision은 딥러닝 모델의 네트워크 레이어 별로 파라미터 단위를 다르게 적용하여 학습 및 추론을 진행하는 방식이다[5]-[8]. 임베디드 시스템에서 딥러닝 모델에 Mixed Precision을 적용하기 위해서는 fp32, fp16, int8등 딥러닝 모델의 연산 변화가 가능한 프레임워크가 필요하다. 본 논문에서는 정확도가 높은 fp32로 학습된 파라미터를 기반으로 Mixed Precision으로 추론연산을 수행하는 딥러닝 네트워크를 프레임워크를 구성하였다. 구성한 프레임워크를 yolov3-tiny[9] CNN 모델에 구현하여 적용하여 보고, fp32와 int8을 혼용해서 사용하는 Mixed Precision에 대해서 추론 연산을 수행하여 모델의 정확도와 연산시간의 관계를 분석하여 보았다.

Ⅱ. 배 경

배경에서는 Mixed Precision 적용을 위해서 필요한 기법인 양자화 기법에 관해서 설명한다. 양자화 방법은 딥러닝 경량화 연구 중 가장 대표적인 방법이다. 양자화 방식은 부동소수점의 IEEE 754 fp32를 고정소수점의 int8로 압축하는 방식이 대표적이다. NVIDIA에서 발표한 논문[10]에서는 Low Bit 양자화에 관하여 설명하고 있는데, 특히 정수형 양자화에 초점을 맞추고 있다.

양자화를 하는 데에 있어 두 가지의 양자화 방법이 있다. Uniform Quantization과 Non-uniform Quantization이다. Uniform Quantization은 Floating point로 된 Weight를 고정소수점(Fixed point) 형태로 변환하는 방식이다. Non-uniform은 가중치에 따라 양자화의 값을 정하는 방식이다. Uniform Quantization은 또 여러 가지 방법으로 나눠지는데 어떤 식으로 양자화를 진행할 것 인지에 따라 달라진다. 여러 식으로 Affine Quantization이 있고 Scale Quantization 등이 있다. Affine Transformation(아핀 변환)의 경우는 파라미터 bit의 비율을 유지한 채 거리만 변화시키는 방식이다[11].

Scale Quantization은 단순히 스케일 변환만으로 범위 매핑을 수행한다[12]. Scale Quantization은 연속적인 실수를 정수로 측정하는 방법이다. Scale Quantization은 주어진 범위 내에서 실수를 정수로 변환하는데, 변환을 수행하기 위해 Scale Factor를 사용합니다.

학습 과정에서의 양자화도 2가지로 나뉜다. PTQ (Post Training Quantization)과 QAT(Quantization Awarness Training) 이다. PTQ 방식은 훈련을 마치고 Export 한 파라미터에 대해 양자화 과정을 거치는 방식이고, QAT 방식은 학습을 수행하는 과정에서 양자화를 진행할 것을 기억하며 학습하는 방법이다. PTQ의 장점으로는 기존 훈련된 모델을 양자화할 수 있다는 점, 원래 모델의 성능을 유지할 수 있다는 점이 있다. QAT의 경우 훈련과정에서 양자화를 고려하기 때문에 양자화된 모델도 정확도 손실을 줄일 수 있다. 또한 양자화 후에도 모델의 성능을 최적화할 수 있다.

본 논문에서는 Mixed Precision 기반 추론 연산을 적용하기 위해서 기존에 학습되어 공개된 fp32 타입의 파라미터에 대해서 추론 수행 시에 양자화/반양자화 연산을 실행하여 파라미터를 생성하고 딥러닝 추론 연산을 수행하는 방식의 Mixed Precision 추론 연산 딥러닝 네트워크를 구현하였다.

III. Mixed Precision 프레임워크

3.1 Mixed Precision 구조

임베디드 시스템에서 Mixed Precision 기반의 인공지능 네트워크 추론 연산을 구현하고 분석하기 위해서 구체적으로 yolov3-tiny모델에 대해서 Mixed Precision을 위한 CNN network 및 연산 방법을 설계 및 구현 적용하였다. 본 논문에서는 yolov3-tiny 네트워크에 Mixed Precision을 적용하는 설정 및 구현 과정을 설명하도록 한다.

Mixed Precision을 네트워크의 각 레이어에 유연하게 적용하는 데 필요한 사항은 각 레이어에서 파라미터에 대한 양자화와 반양자화 연산에 대한 선택과 변환이 이루어지는 동작이 필요하다는 것이다.

그림 1은 Mixed Precision 기반 이미지 추론 동작을 위해서 네트워크 포워딩 시 각 레이어에서 동적으로 양자화 및 반양자화를 수행하는 동작 구조를 도식화한 것이다. 각 레이어는 현재 레이어로 입력값이 들어오면 Mixed Precision을 위한 해당 레이어 설정에 따라서 양자화 및 반양자화를 동반한 네트워크 연산을 수행한다.

Mixed Precision을 수행하기 위해서 각 레이어별 몇 가지 설정은 그림 2에 나타나 있다. 그림 2와 같이 각 레이어의 파라미터 설정에 따른 네트워크 연산을 수행하기 위해서 각 레이어의 연산에 대한 설정 항목에 추가적으로 해당 레이어의 정밀도와 양자화/반양자화 여부에 대한 설정값이 지정된다.

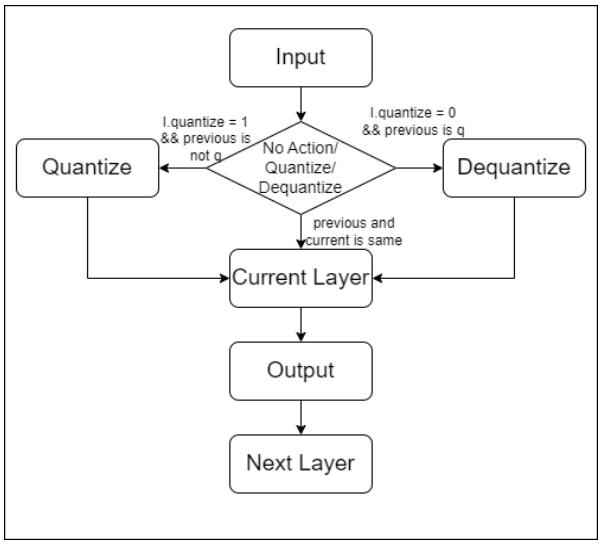


그림 1. Mixed Precision을 위한 각 양자화 및 반양자화 파라미터 network forwarding 구조

Fig. 1. Network forwarding structure of quantized or dequantizated parameter for each laler for Mixed Precision

```
[convolutional]
batch_normalize=1
filters=16
size=3
stride=1
pad=1
activation=leaky
fp32 = 1
fp16 = 0
int8 = 0
quantize = 0
```

그림 2. yolov3-tiny 모델에서 Mixed Precision을 위한 convolutional layer 설정

Fig. 2. Configuration of convolution layer for Mixed Precision in yolov3-tiny model

지정된 설정에 따라 적절하게 입력값의 정밀도를 변환한 후 네트워크 연산을 수행한다. 정밀도 값은 fp32, fp16 혹은 int8이며, 양자화/반양자화 여부는 quantize의 값이 1 또는 0 값에 의해서 정해진다. 그림은 yolov3-tiny의 conv-layer 설정의 예를 보여주고 있으며, 각 레이어의 설정에서 앞서 설명한 fp32, int8등의 옵션 값을 수정하면 그에 맞도록 yolov3-tiny의 네트워크가 구성되어 포워드 연산이 동작한다.

3.2 양자화/반양자화 구현

일반적으로, 신경망 모델은 특정 데이터에 대해서 네트워크를 학습시킨 후, 학습된 파라미터를 특정 파일에 별도로 저장한다. 대부분의 신경망 모델에서 제공하는 파라미터는 fp32로 되어 있으며, 최근 파라미터를 양자화하여 int8의 형태로 제공하는 모델도 증가하고 있다.

본 논문에서는 fp32 형태로 제공되는 파라미터를 이용하여 대해서 CNN 네트워크 연산을 수행할 때 Mixed Precision 연산을 수행하도록 구현하는 것이 목적이다. 즉, 원하는 레이어에 대해서 fp32, fp16 혹은 int8 중에서 원하는 형식의 파라미터값을 이용한 네트워크 연산을 수행하게 하려면, 양자화/반양자화 모듈을 기존 네트워크에 구현하여 주었다. 양자화/반양자화는 기존 Yolo에 적용된 양자화 구현 모델[13][14]을 참고하여 Affine Quantization 모델을 기반으로 구현하였으며, 추가로 오버플로에 대한 데이터 손실을 막기 위해서 Scale Factor를 적용하였으며, int16/int32 크기의 데이터 타입 기반의 임시 버퍼를 활용하였다.

Affine Quantization은 양자화되는 대상 값 중에서 최소값과 최대값의 범위를 찾아내고, 앙자화로 표현되는 값의 범위를 이용해서 서로 간의 비율을 구한다(식 1()). 또한 양자화 대상의 값 및 양자화 후의 값의 최대와 최소를 이용하여 양자화된 값의 중앙값을 계산한다(식 (2)). 이 값을 이용해서 식 (3)과 (4)와 같이 양자화를 수행한다. 그림 3은 양자화 Affine Quantization에 의해서 양자화를 수행하는 코드의 일례를 나타낸 것이다.

$$s = \frac{O_{\max} - O_{\min}}{Q_{\max} - Q_{\min}} \tag{1}$$

$$z = \partial\left(\frac{O_{\max}Q_{\min} - O_{\min}Q_{\max}}{O_{\max} - O_{\min}}\right) \tag{2}$$

$$f(x,s,z) = Clip(\partial\left(\frac{x}{s}\right) + z) \tag{3}$$

$$Clip = \begin{cases} Q_{\min}, (x < Q_{\min}) \\ x_a, \quad (Q_{\max} \le x \le Q_{\max}) \\ Q_{\max}, (x > Q_{\max}) \end{cases} \tag{4}$$

```
int8_t tmp_weights[num];
for(int i= 0; i < num; i++) {
    float x = l.weights[i];
    float s = (MAX, - (MIN)) / (127  -  (-128));
    int8_t = z = 0;

    int8_t tmp= (int8_t)round((x/s)) + z;
    if(tmp< -128) tmp= -128;
    else if(tmp> 127) = 127;

    tmp_weights[i] =tmp
}
```

그림 3. 양자화 수행 소스코드의 일부
Fig. 3. Part of the quantization source code

반양자화는 양자화의 반대로 양자화 값으로부터 기존의 값을 역산하는 방식이다. 이 경우도 마찬가지로 기존의 s와 z를 이용해서 식 (5)와 같이 각 값에 대해서 반양자화 값을 계산한다. 그림 4는 반양자화를 수행하는 코드의 일례를 나타낸 것이다.

$$x = s(x_q - z) \tag{5}$$

```
float tmp_weights[num];
for(int i= 0; i < num; i++) {
    int8_t x = l.weights[i];
    float s = (MAX, - (MIN)) / (127  -  (-128));
    int8_t = z = 0;

    float tmp= s * (x - z);
    tmp_weights[i] = tmp;
}
```

그림 4. 반양자화 수행 소스코드의 일부
Fig. 4. Part of the dequantization source code

3.3 Forward Network 구현

앞서 설명한 대로, Mixed Precision을 사용하여 딥러닝 모델의 추론 연산을 수행하기 위해서 딥러닝 모델은 네트워크의 레이어별로 서로 다른 정밀도 및 데이터 타입으로 설정되어 있다. 딥러닝 네트워크의 각 레이어는 Forward Network 연산을 수행하면서 해당 레이어의 Precision 타입을 검사하여 해당 타입에 필요한 양자화-연산-반양자화-연산을 수행한다. 실제적으로는 네트워크 모델이 기본적으로 fp32 타입의 파라미터와 연산 방식으로 구현되어 있어서, 현재 설정된 네트워크 레이어의 Precision 타입이 fp32가 맞는지 아닌지를 점검하여, fp32가 아니면 양자화/반양자화를 적용한 네트워크 연산을 수행한다. 또한, CNN 모델의 딥러닝 연산 중에서 가장 복잡도가 높은 연산은 컨벌루션 연산이므로, 양자화 및 반양자화를 컨벌루션 연산에 적용한다.

Forward Network에서 양자화-반양자화 연산을 수행하면서 정보 손실을 줄이기 위해서 고려해주어야 하는 부분이 두 가지가 있다. 첫째, 양자화 연산의 결과 변환된 데이터에 대한 저장 범위의 한계로 인해서 발생하는 오버플로를 최소화해야 한다. 둘째, 양자화 데이터에 대한 컨벌루션 연산의 결과에 대한 저장 범위의 한계로 인해서 발생하는 오버플로를 최소화해야 한다. 이를 위해서 구현에서 적용하는 방식은 다음과 같다. 첫째, 양자화 값의 오버플로로 인한 손실을 줄이기 위해서 Scale Factor를 활용한다. 즉, 양자화 과정에서 양자화 대상의 값의 분포도를 계산하여 양자화되는 값의 범위에 대해서 각 값에 곱해져야 하는 Scale값을 계산하여 파라미터의 각 값에 곱해준다. 이를 통해서 양자화의 상대 범위를 지정해줄 수 있음으로써 오버플로를 줄일 수 있다. 둘째, 컨벌루션 연산 시 중간 계산값이 파라미터의 합성곱에 의해서 커짐으로 인해서 발생하는 오버플로를 줄이기 위해서 중간 계산을 저장하는 임시 버퍼의 타입을 크게 지정하여 준다. 예를 들어 int8로 양자화된 값에 대한 합성곱으로 발생할 수 있는 임시 연산 결과의 오버플로를 방지하기 위해서 int32 타입의 임시 버퍼를 사용한다. 그 후, 최종 결과에서는 int8의 타입으로 다시 조정하는 절차를 수행하여 준다.

실제로 yolov3-tiny을 기반으로 Mixed Precision을 적용한 모델은 Jetson Nano나 라즈베리파이와 같은 임베디드 디바이스에서 실행이 가능하나, 특정 시스템에 환경에 따라서는 몇 가지 컴파일러 및 환경 설정 작업이 필요할 수 있다. 일반적으로 대부분의 범용 임베디드 시스템의 구조는 ARM Core 기반 프로세서와 수 GB의 메모리, 임베디드에 최적화된 GPU 등을 포함하고 있다. ARM의 경우 fp32나 int8과 같은 많이 사용되는 데이터 타입에 대한 컴파일 기능을 일반적으로 제공하지만, 특정 시스템에서는 fp16과 같은 데이터 타입의 사용을 위해서는 추가 컴파일러 및 실행환경 설정 작업을 수행해 주어야 한다. 적절한 컴파일러 환경 설정을 통해서, 우리가 구현한 yolov3-tiny 기반 Mixed Precision 모델을 Jetson Nano와 라즈베리파이에서 실행하였을 때 적절히 수행되는 것을 확인할 수 있었다.

Ⅳ. 성능평가

모바일 디바이스, IoT 엣지 디바이스 등과 같은 임베디드 디바이스에서 딥러닝 모델의 연산 오버헤드를 줄이면서 정확도 감소를 최소화하기 위한 방식으로 혼합된 모델 파라미터의 사용과 연산을 적용하는 Mixed Precision 기반 딥러닝 모델의 추론 연산 방법의 구현에 관해서 설명하였다. 임베디드 시스템 환경에서 int8과 fp32 기반 Mixed Precision 딥러닝 모델에 대해서 직접 추론 연산을 수행해 보고 연산 오버헤드와 정확도 등을 분석하기 위해서 성능 평가를 수행하였다. 임베디드 시스템에서 널리 사용되는 대표적인 객체 검출 딥러닝 모델 및 응용 프로그램인 yolo 모델 중에서 C/C++ 모델로 소스가 구현되어 오픈되어있는 yolo-v3[9] 모델 중 yolov3-tiny 모델에 Mixed Precision 방식을 구현 적용하여 성능평가를 실행하였다. 우리는 사전 학습되어 fp32 데이터 타입으로 공개된 yolov3-tiny 모델에 대해서 양자화 및 Mixed Precision을 적용하여 파라미터를 생성 및 적용하였다. 추론 연산과 객체 검출을 위해서 사용한 데이터셋은 ImageNet 데이터셋 중에서 1000개의 이미지를 추출하여 사용하였다.

yolov3-tiny기반 Mixed Precision 모델은 임베디드 디바이스에서 실행이 가능하나, 실제 성능 분석을 위한 데이터 추출은 x86 기반 개인용 컴퓨터에서 우분투 리눅스를 이용하여 상대적 성능을 측정하였다. 성능평가는 크게 두 부분으로 구성되어 있다. 첫째는 Mixed Precision 지원을 위해서 yolov3-tiny 모델의 각 layer에 양자화 연산을 적용하였을 때 발생하는 integer 데이터 타입에 대한 데이터 손실을 분석하였으며, 둘째로, yolov3-tiny 모델에 대해서 레이어 설정을 통해서 int8과 fp32를 혼합해서 적용한 Mixed Precision 기반 추론 연산을 수행했을 때 수행시간 및 정확도에 관한 성능 평가를 실행하였다.

4.1 int8 양자화 오버플로

먼저, fp32값을 int8로 양자화를 하였을 때 발생하는 정보 손실을 분석하기 양자화된 int8 데이터를 이용한 추론 연산 중에 발생하는 값의 오버플로에 대해서 분석하였다. int8로 양자화된 weight와 입력값에 대해서 픽셀 단위 곱셈 연산을 수행했을 때 연산 결과가 int8의 범위 안에 들어가지 않고 오버플로가 나는 비율에 대해서 측정해 보았다. 표 1은 yolov3-tiny 모델에 대해서 layer 1~10에 대해서 단일 픽셀당 weight와 입력을 곱한 결과 중 int8 타입에 대해서 오버플로가 발생한 비율을 나타낸 것이다. ImageNet 데이터셋에 대해서 각각 적용했을 때 각 레이어마다 측정된 오버플로에 대해서 Max와 Min을 측정하였으며, 전체 데이터셋에 대한 평균값은 Avg.로 나타내었다.

결과에서 보는 바와 같이 int8 양자화된 데이터에 대해서 각 픽셀에 대한 weight와 입력값에 대한 곱셈 연산을 수행했을 경우 레이어에 따라서 7~20% 정도의 오버플로가 발생하는 것을 확인할 수 있다. 즉, int8 연산의 결과를 그대로 int8 데이터 타입을 이용하여 저장할 때 해당 비율 정도의 데이터 손실이 발생할 수 있다는 의미이다. 오버플로가 발생할 때 데이터 손실을 발생하므로 딥러닝 연산의 손실이 발생해 심한 경우 객체 검출이 되지 않는 경우도 발생한다. 오버플로를 줄이기 위해서는 연산의 결과를 저장하는 데이터 타입의 크기를 늘려주는 것이 필요하다. 앞서 구현에서 설명한 바와 같이 연산 결과를 임시 int16 데이터 타입에 저장하도록 한 경우, 연산 결과를 분석해보면 오버플로가 전혀 발생하지 않는 것을 확인할 수 있었다.

표 1. 각 레이어별로 int8 양자화 파라미터와 입력에 대한 픽셀 곱 연산의 결과를 int8 데이터에 저장했을 때 발생하는 오버플로 비율(%)
Table 1. Overflow rate(%) that occurs when pixel multiplication is performed on the quantized int8 parameter and input and saved in int8 for each layer

Layer	Avg.	Max	Min
1	9.23	17.02	0.81
2	16.88	29.64	2.33
3	20.26	32.13	6.9
4	20.09	28.92	11.33
5	11.31	18.63	2.29
6	6.93	12.91	2.6
7	10.87	24	1.84
8	7.92	16.29	0.15
9	9.21	13.97	4.39
10	14.36	28.13	1.67

단일 픽셀의 연산과 유사한 방법으로, 각 레이어에서 수행하는 연산 중 가장 복잡도가 높은 합성곱 연산에 대해서 오버플로를 분석하였다. 표 2는 각 레이어별로 int8로 양자화된 weight와 입력값에 대해서 필터 크기에 해당하는 합성곱을 수행하였을 때 발생하는 오버플로의 결과를 나타낸 것이다. 결과에서 보듯이 합성곱 결과를 int16에 저장했을 경우 많은 레이어에서 오버플로 발생이 5% 미만으로 적음을 알 수 있다.

표 2. 각 레이어별로 int8 양자화 파라미터와 입력에 대한 합성곱 연산을int16데이터에 저장했을 때 발생하는 오버플로 비율(%)
Table 2. Overflow rate (%) that occurs when convolution is performed on the quantized int8 parameter and input and saved in int16 for each layer

Layer	Avg.	Layer	Avg.
1	0.03	6	14.62
2	0.07	7	0.5
3	1.47	8	10.13
4	25.35	9	0
5	8.03	10	43.65

그러나 특정 레이어에 대해서는 수십%의 오버플로가 평균적으로 발생하는 것을 알 수 있다. 합성곱의 결과에 대해서는 int16보다 큰 int32에 데이터를 임시 저장하도록 구현하는 것이 필요하다는 것을 확인할 수 있다.

4.2 Mixed Precision 성능 분석

Mixed Precision 추론 연산의 성능을 추론 시간과 정확도 등 두 가지 관점에서 분석하여 보았다. 우리는 먼저 추론 시간은 기존 단일 파라미터 타입을 이용한 추론 연산시간에 비해서 Mixed Precision을 적용하였을 때 추론 시간이 얼마나 변화하는지에 관해서 실험을 진행하였다. 실험은 yolov3-tiny 모델에 대해서 이미지의 객체 추출을 위한 추론 연산을 수행하면서 각 레이어별로 실행한 실행시간 및 이미지 객체 검출 시간을 측정하였다. 실험에서 우리는 먼저 fp32 파라미터만을 이용한 추론을 수행하면서 각 레이어별로 수행시간을 측정한 후, 가장 시간이 오래 걸리는 3개의 레이어인 2, 7, 12에 대해서 Mixed Precision 조합을 만들어서 추론을 수행하였다. 실험에 적용한 파라미터 조합은 fp32_all, fp32_int8(2,7,12), int8_all, int8_fp32(2,7,12)이다. fp32_all은 모든 레이어의 파라미터가 fp32 타입을 나타내며, int8_all은 모든 레이어의 파라미터가 int8 타입을 의미한다. fp32_int8(2,7,12)는 레이어 2,7,12에는 int8 타입의 파라미터이며 이외의 레이어는 fp32를 사용한 것이고, int8_fp32(2,7,12)는 레이어 2,7,12에 대해서 fp32타입이며 나머지는 int8 타입을 사용한 것이다.

그림 5는 네 가지 조합에 대해서 추론 연산을 수행할 때 각 레이어별 수행시간을 측정한 결과를 도식화한 것이다. 그림 5(a)는 절대적인 각 레이어별 절대적인 수행시간을 나타낸 것이며, 5(b)는 상대적인 비율을 나타낸 것이다. 그림 6은 이미지 객체 추출에 걸린 시간을 측정한 것이다. 그림 5에서 볼 수 있는 바와 같이, 각 레이어별로 fp32 타입의 파라미터로 연산을 수행하는 것이 int8 타입으로 연산을 수행하는 것보다 연산시간이 긴 것을 확인할 수 있다. 특히, Mixed Precision을 적용한 경우인 fp32_int8(2,7,12)을 살펴보면, 양자화를 적용한 2, 7, 12 레이어에서 fp32를 적용한 것보다 연산시간이 줄어드는 것을 확인할 수 있다.

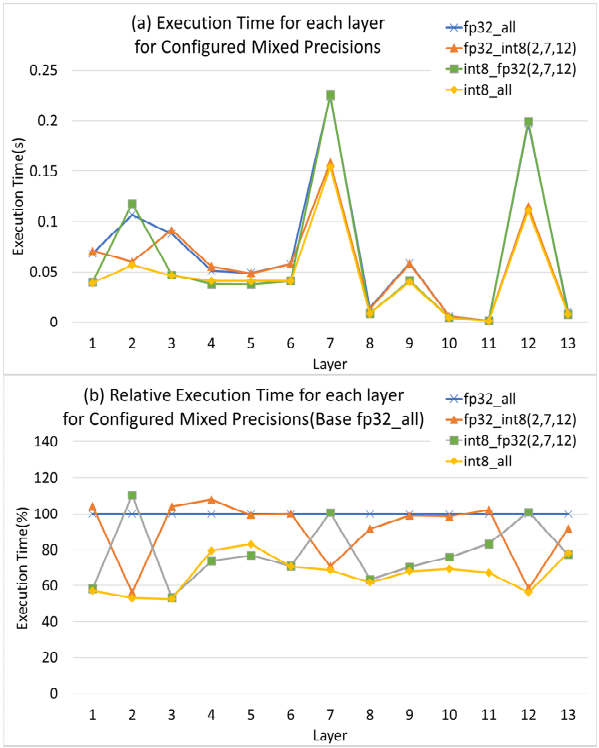


그림 5. yolov3-tiny 모델에 대해서 Mixed Precision 방식을 적용하여 추론 연산을 수행할 때 각 레이어별 수행시간을 측정한 결과

Fig. 5. Result of measuring the execution time for each layer when performing inference operation by applying the Mixed Precision method to the yolov3-tiny model

이러한 수행시간 단축은 전체적인 객체 추출 추론 시간에도 영향을 미치는 것을 그림 6에서 확인할 수 있는데, fp32의 경우가 전체 추론 시간이 가장 길었으며, int8의 경우가 가장 짧은 시간이 걸렸다. 특이한 점은 fp32_int8(2,7,12)가 int8_fp32(2,7,12)보다 추론 시간이 적다는 것인데, 실험환경에 따라서 달라질 수는 있겠지만, 많은 레이어에 대해서 양자화를 수행하는 것보다 연산이 복잡하여 시간이 많이 소모되는 몇몇 레이어에 대해서만 양자화를 적용한 경우가 시간이 더 많이 줄어들 수 있음을 시사한다.

그림 7은 yolov3-tiny 모델에 대해서 Mixed Precision 방식을 적용한 객체 추출 연산의 정확도(Accuracy)에 대해서 실험하여 그 결과를 상대적으로 비교하여 나타낸 것이다.

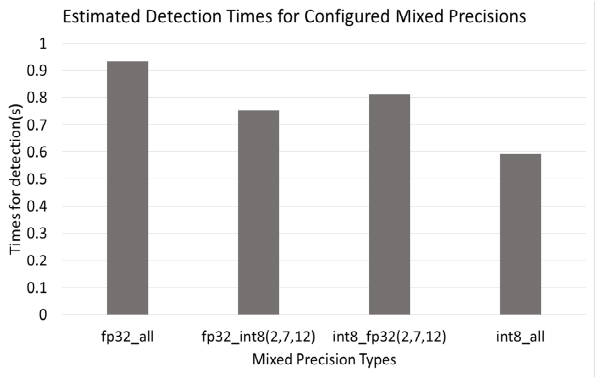


그림 6. yolov3-tiny 모델에 대해서 Mixed Precision 방식을 적용한 객체 추출 추론연산 수행 시간

Fig. 6. Execution time of object detection using Mixed Precision method for yolov3-tiny model

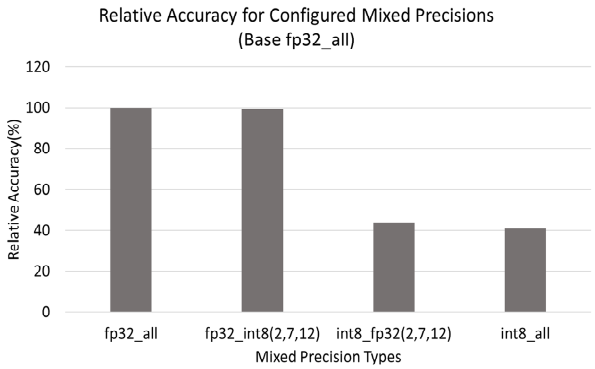


그림 7. yolov3-tiny 모델에 대해서 Mixed Precision 방식을 적용한 객체 추출 정확도 실험 결과

Fig. 7. Experimental results on the accuracy of object detection using the Mixed Precision method for the yolov3-tiny model

즉, fp32의 객체 추출정확도를 100으로 하여, 나머지에 대해서 상대적인 정확도를 나타낸 것이다. 결과에서 보는 바와 같이 fp32의 경우가 int8에 비해서 정확도가 훨씬 높으므로 int8 타입으로 양자화하여 딥러닝 연산을 수행하는 것은 정확도 측면에서 손실이 높다. 그런데, 특정 레이어(2,7,12)에 대해서 양자화를 수행하는 Mixed Precision의 경우 fp32와 정확도 측면에서 손실이 많이 없음을 확인할 수 있다. 그림에서와 같이 fp32_int8(2,7,12)의 경우 fp32와 정확도 성능이 큰 차이가 없음을 확인할 수 있다. 즉, 수행시간이 많이 소모되는 몇몇 레이어에 대해서 양자화 연산을 적용한 Mixed Precision은 정확도의 손실을 줄이면서 추론 연산시간을 단축할 수 있다는 것을 확인할 수 있다.

V. 결론 및 향후 과제

복잡한 인공지능 네트워크 모델로 인한 정확도 증가는 IoT 시스템이나 모바일 장치와 같이 프로세싱 자원이 충분하지 않은 임베디드 시스템에서는 오버헤드가 된다. 임베디드 시스템에서 딥러닝 모델의 연산을 경량화하기 위한 다양한 연구가 진행되었으며 그 중 대표적인 것이 양자화 기법이다. 본 논문에서는 CNN 딥러닝 모델에 Mixed Precision을 적용한 딥러닝 모델을 구현하여 CNN 추론 및 객체 추출에 직접 적용하여 보고 그 성능평가를 하였다. 그 결과, CNN 딥러닝 모델의 레이어별로 연산시간이 많이 소모되는 레이어에는 양자화를 수행한 int8 타입의 파라미터와 연산을 적용하는 Mixed Precision 모델을 적용했을 때, 모든 레이어를 같은 타입으로 실행한 딥러닝 추론 연산보다 효율적인 것을 확인할 수 있다. 특히, fp32 타입으로 수행하는 모델에 비해서 정확도의 손실이 많이 없이 추론 연산시간을 많이 줄일 수 있었다. 향후 과제로써, 다양한 딥러닝 연산에 대해서 여러 가지 Mixed Precision 조합에 대한 적용 및 실제 임베디드 엣지 디바이스에 적용을 통해서 실제적인 활용 방법을 찾아보도록 한다.

References

[1] K. Lee and E. Kim, "Deep learning model lightweight technology analysis", Technical Report, Korea Institute of Science and Technology Information, 2020.

[2] S. Yoo, K.-H. Lee, J. Park, S. J. Yoon, C. Cho, Y. J. Jung, and I. Y. Cho, "Trends in Deep Learning Inference Engines for Embedded Systems", Electronics and Telecommunications Trends, ETRI, Vol. 34, No. 4, pp 23-31, Aug. 2019. https://doi.org/10.22648/ETRI.2019.J.340403.

[3] J. W. Choi, S. Choe, and E. S. Jung, "A Study on the Performance Evaluation of Edge Device according to the Compression Method of Deep Learning Model", Journal of the Institute of

Electronics and Information Engineers, Vol. 58, No. 6, pp. 50-60, Jun. 2021. https://doi.org/10.5573/ieie.2021.58.6.50.

[4] D. Chang, J. Lee, and J. Heo, "Lightweight of ONNX using Quantization-based Model Compression", The Journal of The Institute of Internet, Broadcasting and Communication, Vol. 21, No. 1, pp. 93-98, Feb. 2021. https://doi.org/10.7236/JIIBC.2021.21.1.93.

[5] P. Micikevicius, et al., "Mixed Precision Training", arXiv preprint arXiv:1710.03740, Oct. 2017. https://doi.org/10.48550/arXiv.1710.03740.

[6] D. Dipankar, et al., "Mixed precision training of convolutional neural networks using integer operations", arXiv preprint arXiv:1802.00930, Feb. 2018. https://doi.org/10.48550/arXiv.1802.00930.

[7] J. Wang, S. Fang, X. Wang, J. Ma, T. Wang, and Y. Shan, "High-Performance Mixed-Low-Precision CNN Inference Accelerator on FPGA", in IEEE Micro, Vol. 41, No. 4, pp. 31-38, Jul. 2021. https://doi.org/10.1109/MM.2021.3081735.

[8] C. Latotzke, T. Ciesielski, and T. Gemmeke, "Design of High-Throughput Mixed-Precision CNN Accelerators on FPGA", 32nd IEEE International Conference on Field-Programmable Logic and Applications (FPL), Belfast, United Kingdom, Aug. 2022. https://doi.org/10.1109/FPL57034.2022.00061.

[9] J. Redmon, "YOLO: Real-Time Object Detection", https://pjreddie.com/darknet/yolo/ [accessed: Apr. 01, 2023]

[10] H. Wu, et al., "Integer quantization for deep learning inference: Principles and empirical evaluation", arXiv preprint arXiv:2004.09602, Apr. 2020. https://doi.org/10.48550/arXiv.2004.09602.

[11] J. R. Klauder, "The benefits of affine quantization.", arXiv preprint arXiv:1912.08047, Dec. 2019. https://doi.org/10.48550/arXiv.1912.08047.

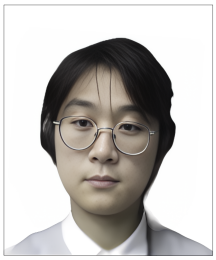
[12] Scale Quantization, https://iq.opengenus.org/scale-quantization/ [accessed: May 01, 2023]

[13] C. Ding, S. Wang, N. Liu, K. Xu, Y. Wang, and Y. Liang, "REQ-YOLO: A Resource-Aware, Efficient Quantization Framework for Object Detection on FPGAs", ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, New York, pp. 33-42, Feb. 2019. https://doi.org/10.1145/3289602.3293904.

[14] Yolov3 Quantization, https://github.com/AlexeyAB/yolo2_light [accessed: May 01, 2023]

저자소개

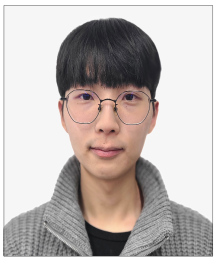
이 종 은 (Jong-Eun Lee)



2017년 3월 ~ 2023년 2월 : 한국외국어대학교 컴퓨터공학부 학사과정

관심분야 : AI, CNN, 딥러닝, 임베디드 시스템

장 경 빈 (Kyung-Bin Jang)



2018년 3월 ~ 현재 : 한국외국어대학교 컴퓨터공학부 학사과정

관심분야 : AI, CNN, 딥러닝, 임베디드 시스템

임 승 호 (Seung-Ho Lim)

2001년 2월 : 한국과학기술원 전기 및 전자공학과(공학사)

2003년 2월 : 한국과학기술원 전기 및 전자공학과(공학석사)

2008년 2월 : 한국과학기술원 전기 및 전자공학과(공학박사)

2008년 3월 ~ 2010년 2월 : 삼성전자 메모리 사업부 책임 연구원

2010년 3월 ~ 현재 : 한국외국어대학교 컴퓨터공학부 교수

관심분야 : 운영체제, 파일 시스템, 임베디드 시스템, DLA for AI, 비휘발성 메모리 시스템, 빅데이터 처리, Hadoop, HDFS