T T A Technical Report

기술보고서

TTAR-xx.xxxx 제정일: 20xx 년 xx 월 xx 일

임베디드 시스템을 위한 콘볼루션 신경망 경량화 지침 (기술보고서)

A Guideline for Lightening Convolutional Neural Networks for Embedded Systems (Technical Report)

TTA 한국정보통신기술협회
Telecommunications Technology Association

기술보고서 초안 검토 위원회　임베디드 소프트웨어 프로젝트그룹(PG601)

기술보고서안 심의 위원회　소프트웨어/콘텐츠 기술위원회(TC6)

	성명	소 속	직위	위원회 및 직위	기술보고서번호
기술보고서(과제) 제안	이대우	(주)알티스트	책임	-	TTAR.xx-xxxx
기술보고서 초안 작성자	이대우	(주)알티스트	책임	-	TTAR.xx-xxxx
사무국 담당	김재웅	TTA	단장	-	TTAR.xx-xxxx
	민선미	TTA	책임		TTAR.xx-xxxx

본 기술보고서 발간 이전에 접수된 지식재산권 확약서 정보는 본 기술보고서의 '부록(지식재산권 확약서 정보)'에 명시하고 있으며, 이후 접수된 지식재산권 확약서는 TTA 웹사이트에서 확인할 수 있습니다.
본 기술보고서와 관련하여 접수된 확약서 외의 지식재산권이 존재할 수 있습니다.

발행인 : 한국정보통신기술협회 회장
발행처 : 한국정보통신기술협회
13591, 경기도 성남시 분당구 분당로 47
Tel : 031-724-0114, Fax : 031-724-0109
발행일 : 20xx.xx..

서 문

1 기술보고서의 목적

이 기술보고서는 자원이 제한적인 임베디드 시스템에서 콘볼루션 신경망 기반 응용을 효과적으로 활용하기 위한 사전 연구 보고서로, 사용하고자 하는 콘볼루션 신경망을 경량화하기 위하여 고려해야 할 지침을 제시한다. 이를 통해 국내 임베디드 소프트웨어 업체들로 하여금, 콘볼루션 신경망 기반 응용의 활용에 대한 진입 장벽을 낮추는 것을 목적으로 한다.

2 주요 내용 요약

콘볼루션 신경망은 영상 처리 등의 분야에서 다양한 응용에 활용되고 있는 대표적인 알고리즘으로, 전통적인 머신러닝 알고리즘에 비해 월등히 우수한 정확성을 제공하지만 연산량과 메모리 요구량이 크다. 따라서, 상대적으로 자원이 제한적인 임베디드 시스템에서 콘볼루션 신경망 기반 응용들을 효과적으로 실행하려면, 콘볼루션 신경망의 경량화 과정이 필수적이다. 본 기술보고서는 먼저 콘볼루션 신경망 경량화 기술 동향을 분석하고, 이를 기반으로 하여 임베디드 시스템을 위한 콘볼루션 신경망 경량화 지침을 제시한다. 이때, 사용하려는 콘볼루션 신경망과 목표 임베디드 시스템은 이미 정해져 있다고 가정한다. 마지막으로, 본 지침을 적용했던 실제 사례를 소개한다.

3 인용 기술보고서와의 비교

3.1 인용 기술보고서와의 관련성

해당 사항 없음.

3.2 인용 기술보고서와 본 기술보고서의 비교표

해당 사항 없음.

Preface

1 Purpose

The technical report is a preliminary report to effectively utilize convolutional neural network based applications in resource-constrained embedded systems, and proposes guidelines for lightening convolutional neural networks. The technical report aims to lower the entry barriers to the use of convolutional neural network based applications by domestic embedded software companies.

2 Summary

Convolutional neural network is a representative algorithm used in various applications in the field of computer vision, audio processing, and so on. It provides excellent accuracy but requires high computational cost. Therefore, to effectively run convolutional neural network based applications on resource-constrained embedded systems, lightening convolutional neural networks is essential. This technical report first analyzes the trend of approaches to lightweight convolutional neural networks, and then proposes a guideline for the lightening procedure for embedded systems. This guideline assumes that both the convolutional neural network to be used and the target embedded system are fixed. Finally, this report introduces an actual case applying the proposed guideline.

3 Relationship to Reference Standards

None.

목 차

임베디드 시스템을 위한 콘볼루션 신경망 경량화 지침

(A Guideline for Lightening Convolutional Neural Networks for Embedded Systems)

1 적용 범위

본 문서는 자원이 제한적인 임베디드 시스템 상에서 콘볼루션 신경망(convolutional neural network) 기반 응용들을 효과적으로 실행하기 위해 필수적인 콘볼루션 신경망 경량화 기술을 다룬다. 여기서 콘볼루션 신경망 경량화란, 주어진 콘볼루션 신경망의 내부 구조를 일부 변경하거나 압축함으로써, 정확성을 조금 희생하더라도 처리 속도와 메모리 요구량을 대폭 개선할 수 있는 기술을 뜻한다. 이러한 기술을 통해 얻을 수 있는 경량화 정도는 대상과 환경에 따라 다르므로, 본 문서는 사용하고자 하는 콘볼루션 신경망과 목표 임베디드 시스템이 이미 정해져 있는 경우를 가정한다. 또한, 기술 성숙도 및 적용 용이성을 고려하여 학습 데이터가 확보된 경우에 적용 가능한 기술만 다루며, 경량화 외의 성능 개선 방안(예: 시스템 최적화 등)에 대해서는 다루지 않는다.

2 인용 표준

해당 사항 없음.

3 용어 정의

3.1 콘볼루션 신경망

콘볼루션 신경망은 머신러닝(machine learning)의 한 유형인 딥러닝(deep learning)에서 많이 사용되는 알고리즘으로, 데이터에서 복잡한 패턴을 찾는데 특히 유용하다. 따라서, 객체 탐지, 얼굴 인식 등의 영상 처리 분야에서 널리 사용되며, 음성과 텍스트에 대해서도 효과적이라고 알려져 있다. 다만, 전통적인 머신러닝 알고리즘에 비해 연산량과 메모리 요구량이 크다.

3.2 텐서

텐서(tensor)는 일반적으로 다차원 배열을 의미한다. 1차원 배열은 벡터(vector), 2차원 배열은 행렬(matrix)이라고 부르나, 3차원 이상까지 고려하는 경우에는 모두 텐서라고 부른다. 신경망에서는 각각의 계층에 대한 입출력 데이터가 텐서로 표현되며,

배치(batch)를 고려하지 않는다면 영상 처리에서는 3차원 텐서가 주로 사용되며, 음성이나 텍스트 처리에서는 2차원 텐서가 주로 사용된다.

4 약어

Conv	(Regular) Convolution
Deconv	Deconvolution (Transposed Convolution)
DWConv	Depthwise Convolution
DWSConv	Depthwise Separable Convolution
GConv	Group Convolution
PConv	Pointwise Convolution

5 콘볼루션 신경망 경량화 기술 동향

연산량이 많은 콘볼루션 신경망을 자원이 제한적인 임베디드 시스템에서 이용하기 위하여 다양한 콘볼루션 신경망 경량화 기술이 개발되고 있다. 콘볼루션 신경망 경량화 기술은 크게 콘볼루션 신경망 구조 경량화 기술과 콘볼루션 신경망 압축 기술로 구분된다.

5.1 콘볼루션 신경망 구조 경량화

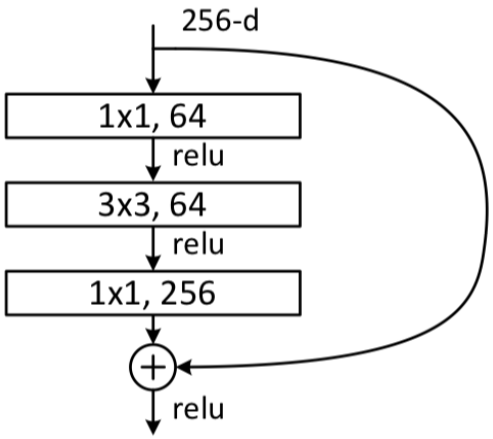
콘볼루션 신경망 구조 경량화 기술이란, 주어진 콘볼루션 신경망에서 연산량이 많이 필요한 계층 또는 모듈을 연산량이 적지만 충분한 표현력을 갖는 것으로 대체하는 방법을 말하며, 주로 콘볼루션 계층의 연산량 감소에 초점을 맞춘다. 또한 현재의 콘볼루션 신경망은 모듈화된 구조로 발전하고 있기 때문에, 경량화 하려는 콘볼루션 신경망에 연산량이 많은 모듈이 존재한다면 이를 효율성이 높은 것으로 대체하는 방법도 고려할 수 있다.

5.1.1 콘볼루션 연산 분해

일반 콘볼루션 연산은 입력 텐서에서 커널(kernel) 차원과 채널(channel) 차원에 해당하는 모든 값을 조합하여 하나의 출력값을 계산해낸다. 그렇기 때문에 콘볼루션 연산의 커널 크기가 크거나 입력 텐서의 채널이 많은 경우, 해당 콘볼루션 계층의 연산량은 크게 증가할 수 있다. 콘볼루션 연산 분해(factorization)란, 이와 같이 많은 연산량을 요구하는 콘볼루션 계층 하나를 적은 연산량의 콘볼루션 계층 여러 개로 대체하는 것을 말한다. 이때, 새로운 콘볼루션 계층을 적절히 설정하면, 전체 신경망의 정확성은 거의 유지하면서도 연산량을 대폭 줄일 수 있게 된다. 콘볼루션 연산 분해는 커널 차원에서의 연산 분해, 채널 차원에서의 연산 분해, 그리고 커널 차원과 채널 차원 간의 연산 분해로 구분된다.

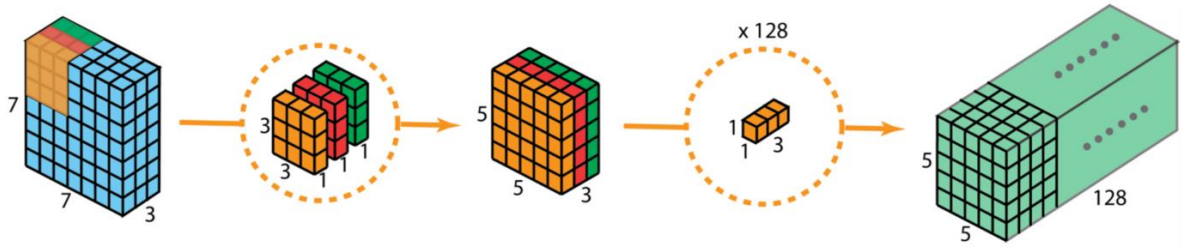
커널 차원에서의 연산 분해는 커널이 큰 콘볼루션 계층 하나를 작은 커널의 콘볼루션 계층 여러 개로 대체하는 방법이다. 영상 처리에서 5×5 Conv 하나를 사용하는 것보다 두 개의 3×3 Conv를 사용하는 것이 더 효과적이라는 사실은 이제 당연하게 여겨지는 커널 차원에서의 연산 분해이다[31]. 이외에 N×N Conv를 1×N Conv와 N×1 Conv로 분해하기도 한다[15, 32].

채널 차원에서의 연산 분해는 입력 텐서의 채널이 많은 경우에 효과적인 방법으로, 병목(bottleneck) 구조[18] 등에서 제안된 바와 같이 연산량을 줄이려는 Conv 앞에 PConv를 추가하는 방법이 대표적이다. 이때, PConv의 출력 텐서의 채널 개수를 적절하게 설정하면, 일종의 특징 차원 축소(feature dimensionality reduction) 효과를 기대할 수 있다. 이러한 효과는 결과적으로 Conv의 입력 텐서의 채널 개수가 축소됨에 따라 연산량이 감소하지만, 정확성 저하는 적을 수 있다. PConv 대신 GConv를 사용하면 연산량이 더욱 감소되나, 정확성 저하가 클 수 있다.



(그림 1) 병목 구조 예시: 입력 텐서를 64 채널로 축소하여 처리 (출처: [18])

커널 차원과 채널 차원 간의 연산 분해는 하나의 콘볼루션 계층을 커널 차원의 정보만 이용하는 콘볼루션 계층과 채널 차원의 정보만 이용하는 콘볼루션 계층으로 분해하는 방법으로, 최근 많이 활용되는 DWSConv가 여기에 해당된다[14, 20, 21, 29, 30, 33]. DWSConv는 DWConv와 PConv의 조합을 의미하며, Conv에 비해 표현력은 유사하면서도 연산량은 상당히 적다. 최근 5×5 DWSConv도 상당히 효과적이라는 결과가 보고되기도 하였다[21, 33].

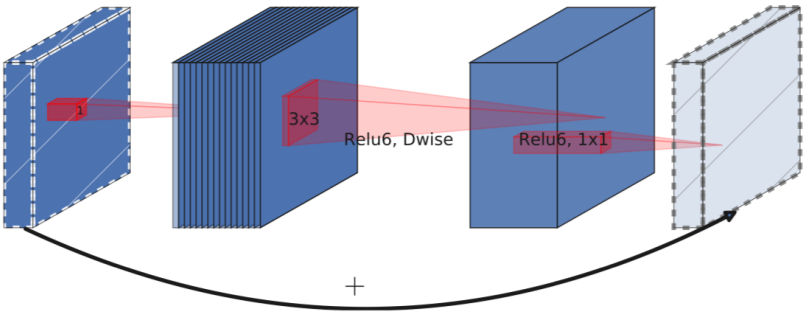


(그림 2) DWSConv의 연산 과정
(출처: https://mc.ai/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning/)

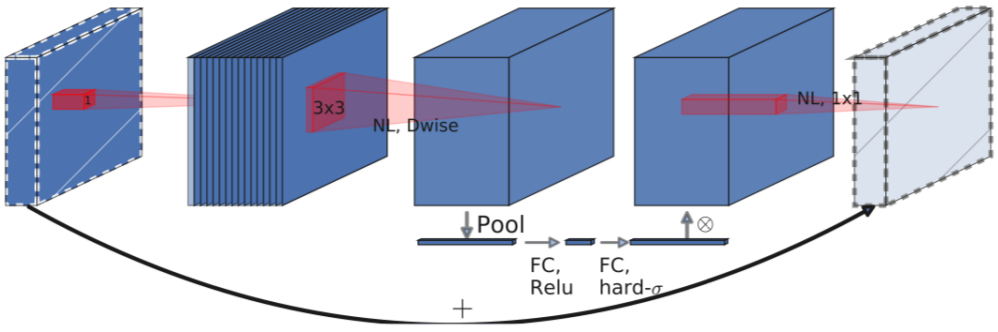
5.1.2 경량 콘볼루션 신경망 구조

최근 콘볼루션 신경망을 모바일 환경이나 임베디드 환경에서 활용하기 위하여, 설계 단계부터 경량화를 최우선으로 하는 연구가 활발히 진행되고 있다. 그렇게 개발된 경량 콘볼루션 신경망은 그 자체를 하나의 모듈로 생각할 수 있다. 실제로 이미지 분류 분야에서 제안된 경량 콘볼루션 신경망을 객체 탐지, 객체 분할 등 다른 영상 처리 분야에서 활용하는 경우가 많다. 이와 같은 경량 콘볼루션 신경망으로 대표적인 것은 모바일넷(MobileNet)[20, 21, 30]과 셔플넷(Shuffle-Net)[26, 36]이다.

모바일넷은 구글(Google)에서 개발한 경량 콘볼루션 신경망으로, 현재 버전3까지 발표되었다. 모바일넷 버전1[20]의 특징은 Conv를 DWConv와 PConv의 조합으로 대체하여 전체 연산량을 대폭 줄인 것이다. 모바일넷 버전2[30]에는 DWSConv 연산을 포함하는 새로운 신경망 모듈인 반전 잔류 블록(inverted residual block)이 적용되었고, 이를 통해 연산량 감소에도 불구하고 처리 속도와 정확성은 오히려 개선되었다. 가장 최근에 개발된 모바일넷 버전3[21]에는 SE(Squeeze-and-Excitation) 모듈[22]이 추가된 반전 잔류 블록이 활용되었으며, 여기에 신경망 구조 검색(network architecture search)[33, 35] 기반의 신경망 최적화 기법이 적용되었다. 그 결과, 버전2에 비해 파라미터 개수는 다소 증가되었으나, 연산량과 처리 속도는 개선되었다.



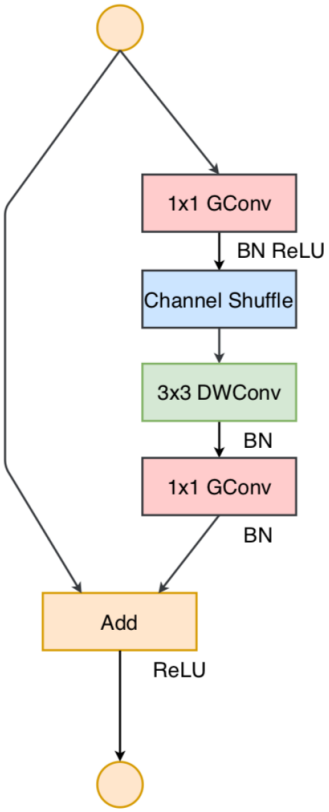
(그림 3) 모바일넷 버전2의 기본 구조: 반전 잔류 블록 (출처: [30])



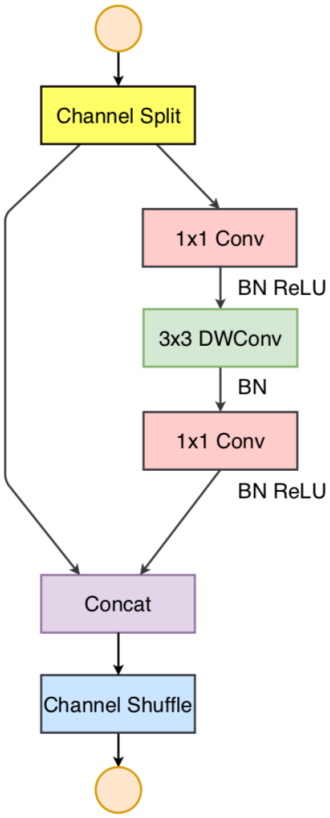
(그림 4) 모바일넷 버전3의 기본 구조: 반전 잔류 블록 + SE 모듈 (출처: [21])

셔플넷은 중국의 스타트업인 메그비(Megvii)가 주도하여 개발한 경량 콘볼루션 신경망으로, 현재 버전2까지 발표되었다. 셔플넷 버전1[36]의 설계 목표는 레즈넷(ResNet)[18]의 핵심 구조인 병목 잔류 블록(bottleneck residual block)의 연산량

감소이다. 이에 따라, 병목 잔류 블록을 구성하는 Conv와 PConv가 각각 DWConv와 GConv로 대체되었고, 연산량 감소에 의한 정확성 저하를 완화하기 위해 채널 셔플(channel shuffle) 연산이 추가되었다. 셔플넷 버전2[26]는 단순히 연산량 감소가 아니라 실제 처리 속도 단축을 목표로 셔플넷 버전1을 개선한 결과이다. 셔플넷 버전2에는 연산의 병렬도를 해치는 GConv 대신 기존의 PConv가 사용되었으며, 병목 잔류 블록에서의 메모리 접근 횟수와 연산량 감소를 위해 채널 분리(channel split) 기법이 적용되었다.



(그림 5) 셔플넷 버전1의 기본 구조
(출처: [26])



(그림 6) 셔플넷 버전2의 기본 구조
(출처: [26])

5.2 콘볼루션 신경망 압축

콘볼루션 신경망 압축 기술이란, 주어진 신경망의 표현력을 가능한 유지하면서 불필요한 파라미터를 제거하거나 파라미터의 정밀도를 축소하여 결과적으로 콘볼루션 신경망의 크기를 줄이는 기술을 말한다. 콘볼루션 신경망 압축 기술을 적용하면 메모리 요구량이 감소하는 동시에, 하드웨어에 따라 연산의 병렬도(degree of parallelism) 증가, 메모리 접근 횟수 감소, 캐시(cache) 효율성 개선 등에 의해 처리 속도 향상과 전력 소모 감소 효과도 얻을 수 있다. 콘볼루션 신경망 압축 기술은 크게 양자화, 정밀도 축소, 가중치 가지치기로 구분된다.

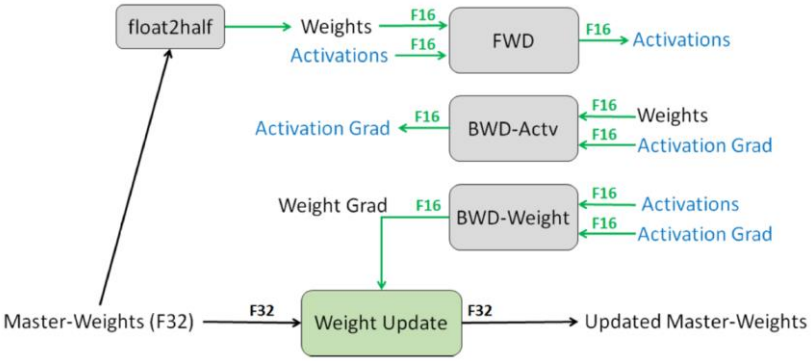
5.2.1 양자화

양자화(Quantization)란, 신경망의 파라미터를 연속값(continuous value)이 아닌 이산값(discrete value)로 표현하는 방식을 말한다. 초기 양자화 기법[17, 23]은 높은 압축률을 얻는 것이 목표였으며, 따라서 처리 속도에서의 이점은 적었다. 하지만 최근에는 대부분의 하드웨어가 부동 소수점 연산보다 정수 연산에 더 최적화되어 있다는 것에 착안하여, 신경망의 연산을 정수 연산으로 대체하는 방향으로 발전하고 있다.

현재 가장 널리 사용되는 양자화 기법은 구글에서 제안한 것으로, 실수(real value) 텐서 간의 연산을 양자화된 정수 텐서 간의 연산으로 근사할 수 있다는 특성을 이용한 기술이다[24]. 이때, 어떤 실수와 양자화된 정수의 관계는 특정 형태의 아핀 변환(affine mapping)으로 정의된다. 따라서, 약간의 데이터 샘플만 있으면 이미 FP32로 학습된 신경망에서도 해당 아핀 변환의 파라미터 근사값을 구할 수 있으며, 이를 학습 후 양자화(post-training quantization) 기법이라고 부른다[8]. 뿐만 아니라, 양자화를 고려하여 신경망을 학습하면 양자화 효과가 증대된다.

5.2.2 정밀도 축소

일반적으로 신경망의 파라미터는 32비트 부동 소수점(floating point) 방식, 즉, FP32로 표현된다. 만약 이를 FP16 또는 FP8로 표현하게 되면 메모리 요구량이 감소될 뿐만 아니라, 하드웨어에 따라 연산 병렬도 증가, 캐시 효율성 개선 등에 의해 처리 속도와 전력 소모가 개선되는 효과도 얻을 수 있다. 하지만 개별 파라미터가 표현할 수 있는 값의 정밀도와 범위가 감소하여 신경망의 표현력에 악영향을 미칠 수 있으므로, 학습 과정에서 이를 보완하기 위한 기술이 필요하다.

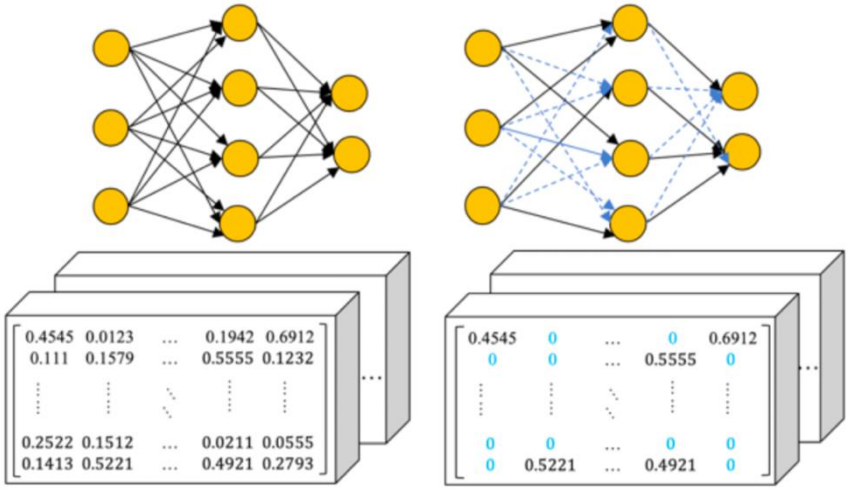


(그림 7) 혼합 정밀도 학습에서의 FP16/FP32 사용 (출처: [27])

가장 대표적인 정밀도 축소 기법은 바이두(Baidu)와 엔비디아(NVIDIA)가 함께 개발한 혼합 정밀도 학습(mixed precision training)[27]으로, FP16의 신경망 파라미터를 얻을 수 있는 신경망 학습 기법이다. 학습 과정에서 파라미터를 FP16 뿐만 아니라 FP32로도 유지하기 때문에 그렇게 명명되었으며, 신경망의 손실값(loss) 비율 조정 등을 통해 FP16 사용에 의한 오차를 최소화하였다.

5.2.3 가중치 가지치기

신경망을 구성하는 파라미터의 특성을 분석하면, 신경망의 최종 결과에 큰 영향을 주지 못하는 파라미터가 존재한다. 예를 들면, 0에 가까운 경우가 그에 해당한다. 이러한 파라미터를 제거하면 신경망의 정확성에는 큰 변화가 없지만, 파라미터를 유지하기 위한 메모리 요구량이 감소된다. 이러한 기법을 가중치 가지치기(weight pruning)[7, 16, 17, 28]라고 부른다.



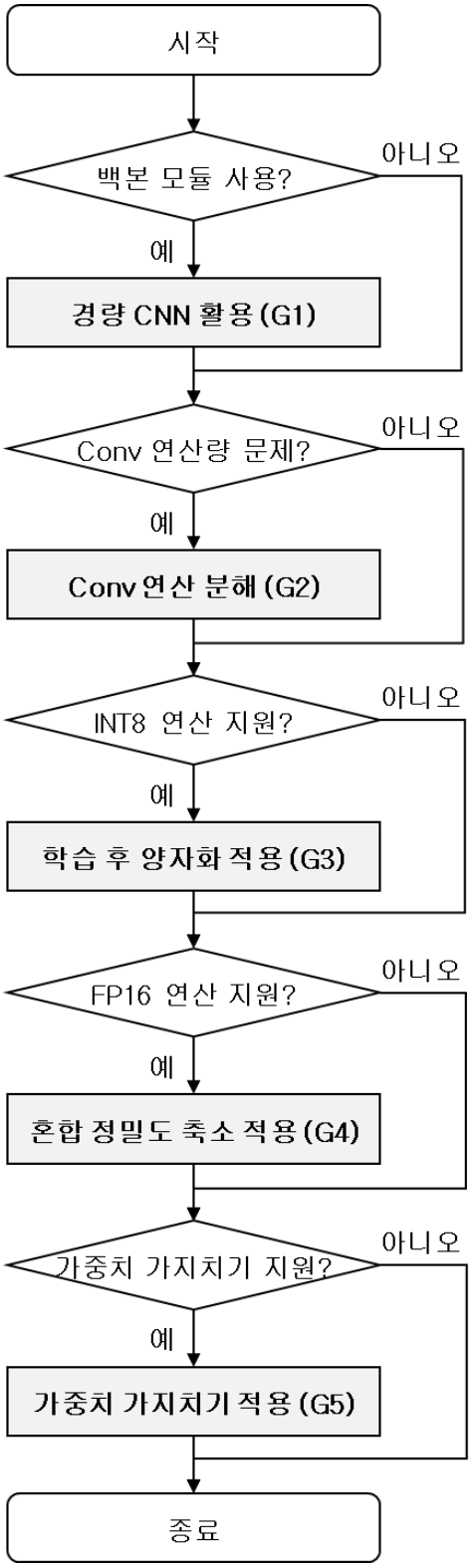
(그림 8) 가중치 가지치기의 예시 (출처: [13])

중요도가 낮은 파라미터를 단순히 전부 제거하면 연산이 불규칙한 구조를 갖게 되어 병렬도가 없어지므로 처리 속도가 크게 저하된다. 불필요한 파라미터를 제거하는 대신 0 으로 대체하면 동일한 결과를 얻을 수 있지만, 이것 역시 처리 속도의 유의미한 향상을 가져오기는 어렵다. 이러한 문제를 해결하기 위하여 연산의 구조적인 특징을 유지하며 가지치기를 적용한 연구들이 진행되고 있으며, 대표적으로는 채널 가지치기(channel pruning)[19, 25, 37]가 있다.

6 임베디드 시스템을 위한 콘볼루션 신경망 경량화 지침

앞에서 소개한 바와 같이 지금까지 다양한 콘볼루션 신경망 경량화 기술이 개발되었고, 지금도 활발히 연구되고 있다. 하지만 이 중에서 업계에 맞는 기술을 특정하는 것은 매우 어렵다. 왜냐하면 업계에서 사용하고자 하는 콘볼루션 신경망과 임베디드 시스템의 특성에 따라, 각각의 경량화 기술이 제공할 수 있는 경량화 정도가 다르기 때문이다. 뿐만 아니라, 요구되는 정확성과 처리 속도, 메모리 요구량 등이 다르므로, 필요한 경량화 정도 역시 달라지게 된다. 따라서, 본 문서에서 제시하는 콘볼루션 신경망 경량화 지침은 사용하고자 하는 콘볼루션 신경망과 목표 임베디드 시스템이 이미 정해져 있는 경우를 가정한다.

본 지침은 그림 9와 같이 각각의 콘볼루션 신경망 경량화 기술을 적용하는 순서와 적용 시에 주의해야 할 사항이 무엇인지에 초점을 맞춘다. 또한, 본 지침은 적용 용이성을 최우선으로 고려하고 있기 때문에 최적의 결과를 보장하는 것은 아니다. 따라서, 경량화 기술 적용 후에는 반드시 필요한 경량화 정도가 달성되었는지 확인해야 한다. 그리고 모든 과정이 종료되기 전에 원하는 수준의 경량화 정도를 달성하는 경우, 이후 과정을 진행할 필요는 없다.



(그림 9) 임베디드 시스템을 위한 콘볼루션 신경망 경량화 지침 적용 과정

6.1 콘볼루션 신경망 구조 경량화 기술 적용 지침

콘볼루션 신경망 구조 경량화 기술을 고려하는 경우, 먼저 콘볼루션 신경망의 백본(backbone) 모듈을 경량 구조로 교체한 다음, 연산량이 많은 콘볼루션 계층에 콘볼루션 연산 분해 기법을 적용하는 것이 좋다.

6.1.1 콘볼루션 신경망의 백본 모듈을 경량 구조로 교체 (G1)

대부분의 콘볼루션 신경망은 효과적인 특징 추출(feature extraction)을 위해 또 다른 콘볼루션 신경망의 일부를 하나의 모듈로 사용하는 경우가 많다. 이와 같이 특징 추출을 위해 사용된 콘볼루션 신경망 모듈을 백본 모듈이라고 부르며, 특히 많은 데이터로 이미 학습된(pre-trained) 콘볼루션 신경망을 백본 모듈로 활용하는 경우가 많다. 영상 처리 분야에서는 주로 이미지넷(ImageNet) 데이터로 학습시킨 이미지 분류용 콘볼루션 신경망을 백본 모듈로 많이 활용한다. 사용하고자 하는 콘볼루션 신경망에도 이러한 백본 모듈이 존재할 가능성이 높다.

만약 사용된 백본 모듈이 정확성을 최우선으로 하여 연산량이 많은 콘볼루션 신경망이라면, 이를 경량 콘볼루션 신경망으로 교체하는 것을 고려해볼 수 있다. 실제로도, 동일한 목적으로 모바일넷이나 셔플넷 등을 사용하여 좋은 효과를 얻은 사례가 많이 존재한다. 다만, 최적의 경량 콘볼루션 신경망은 분야와 환경에 따라 다르므로, 여러 개의 경량 콘볼루션 신경망을 테스트해보는 것을 권장한다.

백본 모듈을 경량 콘볼루션 신경망으로 교체하는 경우, 두 가지를 주의해야 한다. 먼저, 입출력 텐서 크기가 기존 백본 모듈과 일치하는 경량 콘볼루션 신경망을 사용해야 한다. 만약 일치하지 않는 경우에는 Conv나 DWSConv 등을 추가하여 일치하도록 조정해야 한다. 다음으로, 경량 콘볼루션 신경망의 연산들을 목표 임베디드 시스템에서 모두 지원하는지 확인해야 한다. 그렇지 않은 연산을 제거하거나 새로 구현하는 것은 정확성이나 처리 속도 등에 어떤 영향을 가져올지 예상하기 어려우므로, 먼저 다른 경량 콘볼루션 신경망을 고려하는 것이 좋다.

6.1.2 연산량이 많은 콘볼루션 계층에 콘볼루션 연산 분해 기법 적용 (G2)

콘볼루션 신경망에 연산량이 많은 콘볼루션 계층이 존재하면, 해당 계층에 콘볼루션 연산 분해 기법을 적용하는 것을 고려할 수 있다. 연산량이 많은 콘볼루션 계층은 보통 입출력 텐서의 채널이 많은 경우이므로, 채널 차원에서의 연산 분해 기법 또는 커널 차원과 채널 차원 간의 연산 분해 기법을 적용하는 것을 권장한다. 커널 차원에서의 연산 분해는 그리 권장하지 않는데, 왜냐하면 많은 임베디드 시스템에서 특정 크기(예: 3×3)의 커널을 사용하는 콘볼루션 연산이 최적화된 경우가 많기 때문이다.

6.2 콘볼루션 신경망 압축 기술 적용 지침

현재 콘볼루션 신경망 압축 기술에 대해 활발한 연구가 진행되고 있으나, 현재 널리 사용되는 딥러닝 프레임워크(deep learning framework)인 텐서플로우(TensorFlow)[11]나 파이토치(PyTorch)[10] 등에서 공식적으로 지원될 정도로 적용 용이성이 높은 기술은 아직 많지 않다. 따라서 본 지침에 맞게 시도해본 다음에도 추가적인 경량화가 필요한 경우, 최대한 콘볼루션 신경망 구조 경량화 기술로 대응하거나 신경망 또는 목표 임베디드 시스템 변경을 권장한다.

콘볼루션 신경망 압축 기술 중에서는 먼저 학습 후 양자화 기법을 적용하는 것을 권장한다. 왜냐하면 신경망 재학습이 필요하지 않기 때문에 적용하기 쉽기 때문이다. 만약 메모리 요구량이 특히 문제가 되는 경우라면, 그보다 먼저 가중치 가지치기 기법을 적용하는 것이 좋다. 만약 학습 후 양자화 기법의 정확성 저하가 너무 크다면, 혼합 정밀도 학습 기법을 고려한다.

6.2.1 학습 후 양자화 기법 적용 (G3)

텐서플로우는 모델 최적화 툴킷(Model Optimization Toolkit)[12]이라는 확장 패키지를 통해 학습 후 양자화 기법을 지원하며, 파이토치는 글로우(Glow)[6]라는 오픈소스 기술과의 연동을 통해 지원한다. 따라서, 학습 후 양자화 기법을 적용하려는 경우에는 이 두 가지 딥러닝 프레임워크를 활용하는 것을 권장한다.

학습 후 양자화 기법을 적용하는 경우에는 콘볼루션 신경망을 구성하는 연산들이 모두 양자화가 가능한지, 그리고 목표 임베디드 시스템에서도 양자화된 연산을 지원하는지 확인해야 한다. 만약 양자화가 불가능한 연산이 존재하는 경우, 해당 연산 앞뒤로 FP32와 INT8 간의 변환 연산을 추가하는 방법이 있긴 하지만, 대부분의 목표 임베디드 시스템에서는 이를 사용자가 직접 구현해야 한다.

6.2.2 혼합 정밀도 축소 기법 적용 (G4)

현재 엔비디아는 공식적으로 볼타(Volta) 및 튜링(Turing) GPU 상에서 혼합 정밀도 축소 기법을 지원하고 있다. 특히, 텐서플로우와 파이토치, 그리고 MXNet[9]에서는 코드 몇 줄만으로도 혼합 정밀도 축소 기법을 쉽게 적용할 수 있는 자동 혼합 정밀도(automatic mixed precision) 기능을 사용할 수 있다[1]. 텐서플로우와 MXNet는 자동 혼합 정밀도 기능을 지원하며, 파이토치에서는 엔비디아가 제공하는 Apex(A PyTorch Extension)[4]와의 연동을 통해 이를 사용할 수 있다. 따라서, 혼합 정밀도 축소 기법을 적용하려는 경우에는 이 세 가지 딥러닝 프레임워크를 활용하는 것을 권장한다.

혼합 정밀도 축소 기법을 적용하는 경우에도 학습 후 양자화 기법을 적용할 때와 마찬가지로, 목표 임베디드 시스템에서 FP16 연산을 지원하는지 확인해야 한다. 그리고

목표 임베디드 시스템에서 FP16 연산과 FP32 연산의 속도를 비교한 후 혼합 정밀도 축소 기법 적용 여부를 결정하는 것을 권장한다. 왜냐하면 FP16 연산 성능이 최적화되지 않은 경우도 존재하기 때문이다.

6.2.3 가중치 가지치기 기법 적용 (G5)

현재 가중치 가지치기 기법은 텐서플로우의 모델 최적화 툴킷에서 공식적으로 지원된다. 따라서, 가중치 가지치기 기법을 적용하려는 경우, 텐서플로우를 활용하는 것을 권장한다. 다만, 텐서플로우의 가중치 가지치기 기법은 공식 사이트에서도 밝히고 있다시피 처리 속도를 개선하지는 못하므로, 메모리 요구량이 문제가 되는 경우에만 고려하는 것이 좋다.

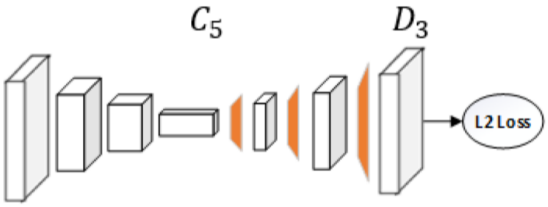
학습 후 양자화 기법과 혼합 정밀도 축소 기법과는 달리, 가중치 가지치기 기법은 학습 시작 전에 압축률을 미리 설정해야 한다. 일반적으로 압축률이 지나치게 높으면 정확도가 크게 감소하므로, 반복 실험을 통해 적절한 값으로 설정한다.

7 적용 사례

본 문서에서 제안하는 콘볼루션 신경망 경량화 지침의 효과를 간단히 소개하기 위하여, 실제 적용 사례를 소개한다. 본 사례의 목표는 그림 9와 같이 드론 조종을 위한 동작 인식 응용 개발로, 드론에 탑재된 카메라로부터 수집된 영상에서 사용자의 자세 또는 동작을 분석하여, 특정 패턴이 인지되면 그에 따라 드론의 비행 상태를 변경하는 것이다.

(그림 10) 드론 조종을 위한 동작 인식 응용 예시

영상에서의 사용자 자세 또는 동작을 분석하기 위하여, 본 사례에서는 마이크로소프트(Microsoft)에서 개발한 포즈레즈넷(PoseResNet)[34]을 경량화하였다. 포즈레즈넷은 백본 모듈로 레즈넷-50을 이용하며, 여기에 3개의 Deconv와 1개의 PConv를 차례로 연결하여 최종 결과를 얻는다. 모델 구현 및 학습은 텐서플로우를 이용하였으며, 마이크로소프트에서 공개한 파이토치 기반 코드[5]와 동일하게 구현하였으나, 자체 수집한 데이터를 사용하였다.



(그림 11) 좌: 포즈레즈넷의 구조(출처: [34]), 우: 자세 추정의 예시(출처: Pexels)

목표 임베디드 시스템의 하드웨어는 Firefly RK3399[2]로, CPU는 2개의 ARM Cortex-A72 코어와 4개의 ARM Cortex-A53 코어로 구성되며, ARM Mali-T864 GPU와 2GB DDR3 메모리가 탑재되었다. 목표 임베디드 시스템의 소프트웨어는 Ubuntu Linux 18.04와 ARM Compute Library 19.05[3]로 구성되며, OpenCL을 이용하여 콘볼루션 신경망을 실행하였다.

본 사례에서는 위와 같이 사용하려는 콘볼루션 신경망과 목표 임베디드 시스템이 정해진 상태에서, 본 문서에서 제안하는 콘볼루션 신경망 경량화 지침을 적용하였다. 다만, 적용 용이성을 판단하여 불필요하다고 판단되는 지침은 제외하였다. 적용된 각각의 지침에 대한 설명과 적용 결과는 다음과 같다.

G1 적용: 백본 모듈을 모바일넷 버전2로 교체
G2 적용: 모든 Deconv 앞에 PConv 추가하여, Deconv 입력 텐서를 64 채널로 축소
G3 미적용: 텐서플로우에서 Deconv의 양자화를 지원하지 않음
G4 적용: 혼합 정밀도 축소 기법 적용
G5 미적용: ARM Compute Library에서 가중치 가지치기 기법을 지원하지 않음

<표 1> 본 문서에서 제안한 콘볼루션 신경망 경량화 지침의 적용 사례

경량화 수준	정확도(PCKh@0.5)	초기화 시간(sec)	처리 시간(sec/image)
미적용	81.4	12.113	1.054
G1	77.7	5.878	0.704
G1+G2	76.3	6.062	0.354
G1+G2+G4	76.0	5.929	0.352

경량화 전후를 비교하면, 정확도는 다소 저하되었지만 실제 응용 수준에서 지장을 주지 않을 정도였으며, 대신 처리 시간이 1/3 정도로 감소하였다.

부 록 Ⅰ-1

(본 부록은 기술보고서를 보충하기 위한 내용으로 기술보고서의 일부는 아님)

지식재산권 확약서 정보

해당 사항 없음.

부 록 Ⅰ-2

(본 부록은 기술보고서를 보충하기 위한 내용으로 기술보고서의 일부는 아님)

시험인증 관련 사항

Ⅰ-2.1 시험인증 대상 여부

해당 사항 없음.

Ⅰ-2.2 시험표준 제정 현황

해당 사항 없음.

부 록 Ⅰ-3

(본 부록은 기술보고서를 보충하기 위한 내용으로 기술보고서의 일부는 아님)

본 기술보고서의 연계(family) 기술보고서

해당 사항 없음.

부 록 Ⅰ-4

(본 부록은 기술보고서를 보충하기 위한 내용으로 기술보고서의 일부는 아님)

참고 문헌

[1] https://developer.nvidia.com/automatic-mixed-precision
[2] https://en.t-firefly.com/product/rk3399.html
[3] https://github.com/arm-software/ComputeLibrary
[4] https://github.com/NVIDIA/apex
[5] https://github.com/microsoft/human-pose-estimation.pytorch
[6] https://github.com/pytorch/glow
[7] https://medium.com/tensorflow/tensorflow-model-optimization-toolkit-pruning-api-42cac9157a6a
[8] https://medium.com/tensorflow/tensorflow-model-optimization-toolkit-post-training-integer-quantization-b4964a1ea9ba
[9] https://mxnet.apache.org/
[10] https://pytorch.org/
[11] https://www.tensorflow.org/
[12] https://www.tensorflow.org/model_optimization
[13] 이용주 외 "경량 딥러닝 기술 동향", 전자통신동향분석 34권 2호, 2019.
[14] F. Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions," CVPR, 2017.
[15] I. Freeman et al. "EffNet: An Efficient Structure for Convolutional Neural Networks," ICIP, 2018.
[16] S. Han et al. "Learning both Weights and Connections for Efficient Neural Networks," NIPS, 2015.
[17] S. Han et al. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," ICLR, 2016.
[18] K. He et al. "Deep Residual Learning for Image Recognition," CVPR, 2016.
[19] Y. He et al. "Channel Pruning for Accelerating Very Deep Neural Networks," ICCV, 2017.
[20] A. G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv:1704.04861, 2017.
[21] A. Howard et al. "Searching for MobileNetV3," arXiv:1905.02244, 2019.
[22] J. Hu et al. "Squeeze-and-Excitation Networks," CVPR, 2018.
[23] I. Hubara et al. "Binarized Neural Networks," NIPS, 2016.
[24] B. Jacob et al. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," CVPR, 2018.
[25] J.-H. Luo et al. "ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression," ICCV, 2017.

[26] N. Ma et al. "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design," ECCV, 2018.
[27] P. Micikevicius et al. "Mixed Precision Training," ICLR, 2018.
[28] P. Molchanov et al. "Pruning Convolutional Neural Networks for Resource Efficient Inference," ICLR, 2017.
[29] L. Kaiser et al. "Depthwise Separable Convolutions for Neural Machine Translation," ICLR, 2018.
[30] M. Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks," CVPR, 2018.
[31] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition," ICLR, 2015.
[32] C. Szegedy et al. "Rethinking the Inception Architecture for Computer Vision," CVPR, 2016.
[33] M. Tan et al. "MnasNet: Platform-Aware Neural Architecture Search for Mobile," CVPR, 2019.
[34] B. Xiao et al. "Simple Baselines for Human Pose Estimation and Tracking," ECCV, 2018.
[35] T. Yang et al. "NetAdapt: Platform-aware Neural Network Adaptation for Mobile Applications," ECCV, 2018.
[36] X. Zhang et al. "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," CVPR, 2018.
[37] Z. Zhuang et al. "Discrimination-aware Channel Pruning for Deep Neural Networks," NeurIPS, 2018.

부 록 Ⅰ-5

(본 부록은 기술보고서를 보충하기 위한 내용으로 기술보고서의 일부는 아님)

영문기술보고서 해설서

해당 사항 없음.

부 록 Ⅰ-6

(본 부록은 기술보고서를 보충하기 위한 내용으로 기술보고서의 일부는 아님)

기술보고서의 이력

판수	채택일	기술보고서번호	내용	담당 위원회
제1판	20xx.xx.xx	제정 TTAR-xx.xxxx	-	임베디드 소프트웨어 PG (PG601)