

기술조사보고서

딥러닝 모델 경량화 기술 분석

2020. 11.

지능형인프라기술연구단

제1장 서론 1

| | |
|------------------------------------|---|
| 제 1 절 딥러닝 모델 경량화 기술 배경 및 필요성 | 1 |
| 1. 연구의 배경 | 1 |
| 2. 연구의 필요성 | 2 |

제2장 Quantization 기반 경량화 기술 6

| | |
|--------------------------------|----|
| 제 1 절 Quantization 기술 개요 | 6 |
| 1. 딥 러닝에서의 학습 과정 개요 | 6 |
| 2. CNN 모델에서의 학습 과정 개요 | 11 |
| 3. RNN 모델에서의 학습 과정 개요 | 22 |
| 4. 모델 경량화 기술 개요 | 25 |
| 제 2 절 Quantization 기법 | 28 |
| 1. 기술의 개요 | 28 |
| 2. 이진화 기법 | 30 |
| 3. k-비트 양자화 기법 | 32 |
| 4. Bi-Real Network | 34 |
| 제 3 절 지식 증류 기법 | 37 |
| 1. 기술의 개요 | 37 |
| 2. Kill-the-bits | 38 |
| 제 4 절 하이브리드 및 기타 기법 | 42 |
| 1. MetaQuant | 42 |

제3장 경량 네트워크 구조 기법 47

| | |
|--|----|
| 제 1 절 CNN 모델의 일반적 구조 | 47 |
| 1. CNN 모델의 역사적 배경 | 47 |
| 2. CNN 모델의 일반적인 연산 기법 | 48 |
| 3. CNN 모델 연산의 특징 | 49 |
| 제 2 절 CNN 모델에 대한 경량 네트워크 구조 | 50 |
| 1. Residual Connection, Bottleneck | 51 |
| 2. Dilated Convolution | 52 |
| 3. Depthwise Seperable Pointwise Convolution | 53 |
| 4. Grouped Convolution | 55 |
| 5. Dense Convolution | 55 |
| 6. Compound Scaling | 56 |
| 7. Shift Convolution | 58 |
| 8. Searching for Mobilenet3 | 63 |

제4장 모델 경량화 연구 분야에 관한 분석 66

| | |
|----------------------------------|----|
| 제 1 절 모델 경량화 기술의 평가를 위한 척도 | 66 |
| 제 2 절 현재 모델 경량화 기술의 한계 | 68 |
| 제 3 절 적절한 모델 경량화 기술의 선택 방안 | 70 |

제5장 모델 경량화 기술의 활용 방안과 전망 72

| | |
|-----------------------------|----|
| 제 1 절 활용 방안 및 기대효과 | 72 |
| 1. 활용 방안 | 72 |
| 2. 기대효과 | 73 |
| 제 2 절 모델 경량화 기술 발전 전망 | 73 |

CONTENTS

그림목차

| | |
|--|----|
| [그림 1-1] CNN 모델 크기와 정확도 | 3 |
| [그림 1-2] 연두에 따른 NVIDIA의 GPU 목록과 성능 | 5 |
| [그림 2-1] Overview of Supervised Machine Learning | 6 |
| [그림 2-2] Human-In-The-Loop process in Deep Learning | 7 |
| [그림 2-3] 커널의 종류와 동작 결과 예시 | 12 |
| [그림 2-4] 일반적인 인공신경망의 구조 | 13 |
| [그림 2-5] CNN의 구조 | 14 |
| [그림 2-6] 컬러 이미지 데이터에 대한 텐서 표현 | 14 |
| [그림 2-7] 하나의 채널에 대한 합성곱 계층의 동작 | 15 |
| [그림 2-8] 스트라이드가 1로 설정된 경우 (좌)와 2로 설정된 경우 (우). | 15 |
| [그림 2-9] 패딩이 적용되지 않은 합성곱 연산 (좌)과 zero-padding이 적용된 합성곱 연산 (우). | 16 |
| [그림 2-10] Max-pooling 기반 풀링 계층의 동작. | 17 |
| [그림 2-11] FNN을 이용하여 이미지를 처리하기 위한 이미지 벡터화 | 18 |
| [그림 2-12] Recurrent Neural Network과 Feed-Forward Neural Network 구조의 차이 | 22 |
| [그림 2-13] recurrent neural network | 23 |
| [그림 2-14] 가지치기 기법 개요 | 26 |
| [그림 2-15] 파라미터의 양자화 예시 | 29 |
| [그림 2-16] signum 함수와 STE | 32 |
| [그림 2-17] 뉴럴 네트워크 구조에 따른 파라미터 수와 정확도의 변천 | 33 |
| [그림 2-18] LQ-Net에서의 기준 벡터와 양자화 된 값의 예 (k=2인 경우) | 34 |
| [그림 2-19] feature map의 값의 표현 종류(D, different value)에 따른 표현 범위 | 35 |
| [그림 2-20] (a) Sign함수와 그 미분 값, (b) Clip 함수와 그 미분 값, (c) Approximate Sign 함수와 그 미분값. | 35 |
| [그림 2-21] Representation Capability 증강을 위해 BatchNorm수행 후 바로 identity connection을 5 | |

| | |
|--|----|
| 행하도록 제안함 | 35 |
| [그림 2-22] 유사한 부류의 분류 (in-domain classification) 성능 향상에 긍정적인 영향을 미치는 soft label방법 | 37 |
| [그림 2-23] imagenet data기준 ResNet-18, ResNet-50모델에 대해 kill-the bits 모델의 압축률과 분류 정확도 성능 비교 | 39 |
| [그림 2-24] 2D data에 대한 0-k means기반 2종류의 quantization 예시 (Norouzi, 2013) | 39 |
| [그림 2-25] 4D data에 대한 Cartesian quantization예시 | 35 |
| [그림 2-26] Meta Quantizer 동작 개괄 | 43 |
| [그림 2-27] meta quantizer를 quantization 학습 과정 중에 도입 | 44 |
| [그림 3-1] CNN 모델 구조 | 48 |
| [그림 3-2] Convolution연산의 Filter와 pooling 의 예시 | 49 |
| [그림 3-3] ResNet, WideResNet 그리고 PyramidNet | 49 |
| [그림 3-4] Pruning의 예시 | 50 |
| [그림 3-5] parameter수와 accuracy를 고려한 Neural Network Architecture의 변화 | 51 |
| [그림 3-6] Plain Network과 Residual block | 51 |
| [그림 3-7] all-3x3 bottleneck(for ResNet50/101/152) | 52 |
| [그림 3-8] Dilated Convolution (Atrous Convolution) (Fu, 2017) | 53 |
| [그림 3-9] DeepLab3에서 이용한 Dilated Convolution (Chen, 2018) | 53 |
| [그림 3-10] DepthWise Separable Point-Wise Convolutions | 54 |
| [그림 3-11] Group Convolution | 55 |
| [그림 3-12] (위) : 음영위주의 커널 학습 그룹, (아래) : 색상과 패턴위주의 학습 그룹 | 55 |
| [그림 3-13] DenseNet | 56 |
| [그림 3-14] ResNet과 DenseNet (additive구성 vs. concat 구성) | 56 |
| [그림 3-15] depth, width, resolution 확장하면서 효율적으로 모델 탐색 | 57 |
| [그림 3-16] (좌) 모델 크기 vs. imageNet 정확도 (우) FLOPS vs. imageNet 정확도 | 57 |
| [그림 3-17] Shift 연산 후 1x1 point-wise convolution으로 구성한 ShiftNet | 58 |
| [그림 3-18] group shift operation | 58 |
| [그림 3-19] Active Shift operation | 59 |

| | |
|---|----|
| [그림 3-20] Active Shift operation의 shift 값이 정수가 아닌 경우 bilinear interpolation | 59 |
| [그림 3-21] Group Shift, Active Shift 그리고 Sparse Shift | 60 |
| [그림 3-22] Residual Block 과 Inverted Residual Block | 61 |
| [그림 3-23] FE-Block의 구조와 FE-BLOCK을 이용한 Shift Network의 구조 | 62 |
| [그림 3-24] Sparse learning 전후의 ShiftResNet20의 Block2_2 shift position 분포 76% shift map구성된 ShiftResNet20의 Block2_2 | 63 |
| [그림 3-25] Sigmoid, hard-sigmoid, Swish, hard-Swish 함수 | 64 |
| [그림 3-26] RELU6 함수 | 64 |
| [그림 3-27] 마지막 network 계층의 수정, 이를 통해 latency 성능 11% 향상 | 65 |
| [그림 4-1] 효율성 평가 척도 왼편부터 (1) 추론 시간 대비 accuracy (shiftNet), (2) Latency 대비 accuracy (mobilenet-v3), (3) 학습 parameter수 대비 accuracy(EfficientNet) | 66 |
| [그림 4-2] (좌) Top1 accuracy, (중) Information Density, (우) NetScore, 빨간색 - SqueezeNet [Iandola, 2016a] 및 SqueezeNext [Gholami, 2018a] 모델 계열 | 67 |

CONTENTS

표목차

| | |
|---|----|
| <표 1-1> 모델의 정확도와 학습시간 증가 | 4 |
| <표 2-1> 모델 경량화 기술의 분류. | 25 |
| <표 2-2> 양자화 대상에 따른 분류 | 30 |
| <표 3-1> AlexNet 각 계층의 구성 | 48 |
| <표 3-2> parameter수와 accuracy를 고려한 Neural Network Architecture의 특징. | 51 |
| <표 3-3> bottleneck 모델 적용에 따른 총 연산량 감소 | 54 |
| <표 3-4> Depthwise Separable PointWise Convo.에 따른 연산, parameter수 감소 계산 | 54 |
| <표 4-1> EfficientNet의 Baseline model (mNASNet으로 구현된 최적화 모델). | 70 |

제 1 절 딥 러닝 모델 경량화 기술 배경 및 필요성

1. 연구의 배경

□ 딥 러닝 모델의 각광 및 이용 분야 증가

- 최근 딥러닝 모델은 기존의 통계적 기법을 포함한 기존의 여러 기법 대비 훨씬 높은 정확도를 제공함에 따라 무수히 많은 분야에서 빠르게 도입, 활용되고 있음
- 특히 컴퓨터 비전 및 자연어 처리, 음성 인식, 언어 번역, 자율 주행, 로봇 제어 등의 분야에서는 기존의 통계적 기법을 모두 대체할 정도로 압도적인 정확도로 인해 이들 분야의 최근 연구들은 모두 딥 러닝에 기초한다고 볼 수 있을 정도임
- 더불어, 최근 딥 러닝 모델은 모바일 기기, IoT 기기 및 엣지 컴퓨팅(edge computing)을 위해 저전력, 저성능의 제약을 가지는 소형 전자기기에서도 동작할 수 있도록 보다 소형화되고, 추론을 위한 요구 연산 성능 및 요구 전력을 최소화시키기 위한 노력들이 기업체 중심으로 많이 연구되고 있음

□ 모델 학습의 시간 및 비용의 큰 증가

- 딥 러닝 모델을 포함한 지도 방식(supervised learning)의 기계학습 모델의 개발 과정은 대량의 레이블된 데이터(labeled data)를 이용하여 모델에 입력값과 이에 대해 모델이 출력해야할 정답을 함께 학습하는 과정이 반드시 요구됨
 - 이러한 모델 학습 과정은 여러 단계로 구성되며, 사용자가 요구하는 특정 정확도에 도달하기 전까지 시행착오를 통해 각 단계에서의 작업을 조절해가며 반복적으로 수행된다는 특징을 가짐
 - 또한 매 단계마다 개발자의 개입이 요구되는 일괄 작업(batch job)으로 수행

되므로, 기계 학습에서의 모델 학습은 일련의 Human-In-The-Loop(HITL) 프로세스로 정의하고 있음

- 이러한 딥 러닝 모델은 학습과 추론 과정에서 고비용의 전산 자원과 함께 많은 연산량을 필요로 함

■ 모델 크기의 증가에 따라 연산량 또한 비례하여 증가하고 있음

- 또한 그에 따른 전력 소모량 또한 크고, 많은 학습 시간을 필요로 함에 따라 개발 지연 등의 문제를 노출해 왔음
- 이러한 문제를 해결하기 위한 여러 방법들(예, 분산 학습 기법[Ben-Nun2019]이나 TPU[Jouppi2017] 등)이 존재하나 주로 하드웨어적인 개선 방법들이 주류를 이루어 왔음

□ 모바일 기기 등 저성능 계산 장치를 위한 딥 러닝 모델 배포 및 활용 필요성의 급속한 증가

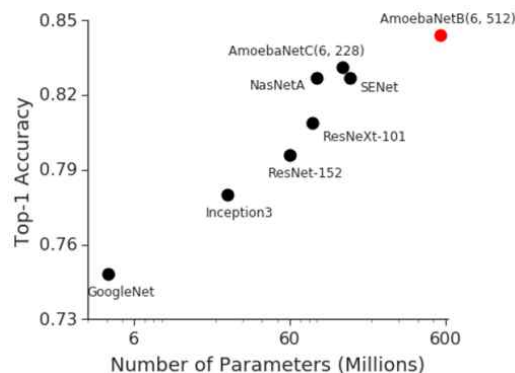
- 최근의 드론, 로봇을 포함하여, 스마트폰 및 다양한 IoT 기기들은 그 컴퓨팅 환경의 특성으로 인해 제한된 연산 능력, 메모리, 전력을 가지고 효율적으로 컴퓨터 비전, 음성 및 자연어 처리 등과 관련한 딥 러닝 모델 탑재 및 추론이 지속적으로 요구되고 있음
- 이를 위해 딥 러닝 알고리즘을 경량화 하여 적은 저장 공간, 연산량을 요구하면서 고속으로 추론할 수 있는 기술을 필요로 함
- 이에 따라 딥 러닝 모델의 경량화 및 추론에서의 효율성에 관한 관심이 점차 증가하고 있음

2. 연구의 필요성

□ 딥 러닝 모델의 크기 증가

- 딥 러닝 모델의 크기 면에서 살펴보자면, 그림 1-1에서 보이는 바와 같이 이미지 인식, 분류 등에 많이 사용되는 모델인 CNN(Convolutional Neural Network)만 살펴보더라도, 정확도 향상과 모델의 크기 간에는 주된 상관관계(correlation)가 있음을 확인할 수 있음

- 즉, 높은 정확도를 위해서 지속적으로 모델의 크기가 커지는 경향을 보임
- ILSVRC의 2014년 우승 모델인 GoogleNet[Szegedy2015]의 경우 약 4백만 개의 파라미터로 74.8%의 top-1 정확도를 보인 반면, 2017년의 우승 모델인 Squeeze-and-Excitation Network[Hu2018a]은 145.8 백만 개의 파라미터로 82.7%의 top-1 정확도를 보임. 정리하면, CNN 모델의 경우 3년 동안 7.9%의 정확도 향상이란 진보를 이룬 반면, 파라미터 수는 약 36 배 이상 폭발적으로 증가함
- 이러한 현상은 NLP 분야에서도 동일하게 목격되고 있음. 2019년 6월에 공개된 XLNet[Yang2019a]이 340M 개의 파라미터로 구성되는데, 같은 해 12월에 공개된 GPipe[Huang2019a]는 556M 개의 파라미터를 가짐
- 또한 올해 OpenAI에서 공개한 GPT-3[Brown2020a]는 175,000M 개의 파라미터를 가지는 것으로 보고하고 있음



[그림 1-1] CNN 모델 크기와 정확도
[Canziani, 2017]

□ 딥 러닝 모델의 연산 필요량의 증가

- 연산량의 증가 측면에서 보자면, 2012년 AlexNet[Alex2012a, Krizhevsky2017a] 모델이 1GFLOPS 미만의 연산량을 보였던 반면, 2018년 NASNET-A-Large[Zoph 2017b] 모델은 24 GFLOPS 이상의 연산량으로 약 25배 이상 증가하였음 [Bianco2018a]
- 이는 '24개월마다 반도체 집적회로의 성능이 2배로 증가한다'는 무어의 법칙

[Waldrop2016a]보다도 딥 러닝 모델의 추론에 있어 요구 연산량의 증가 속도가 훨씬 빠르게 커지고 있음을 시사

- 모델 학습 및 추론에 요구되는 연산량은 동일 구조의 모델이라 하더라도 모델의 깊이가 깊어짐에 따라 파라미터 개수가 증가하면서 같이 증가하는 경향을 보임
- 표 1은 ILSVRC의 2015년 사람의 사물 분류 인지 오차 능력으로 알려진 ResNet모델[He2016a, He2016b]의 계층 깊이의 증가에 따른 정확도 향상과 이에 따른 학습 시간 증가를 나타냄¹⁾
- 정확한 연산을 위해서는 막대한 연산량 증가가 요구되고, 이에 비례하여 학습 시간도 늘어남.
- 학습 시간의 증가는 에너지 면에서 또한 문제가 됨. 학습 시간의 증가는 학습에 요구되는 전산자원이 필요로하는 전력을 증대시킴. 특히 최근 모델들은 많은 수의 파라미터로 인해 더더욱 많은 연산량과 학습 시간을 필요로 함
 - Hao 등[Hao2019a]은 NLP를 위한 딥 러닝 모델 중 하나인 트랜스포머(transformer) 모델을 학습하여 그 결과물을 얻는 과정에서 쓰이는 전력 생산에 대한 탄소 소비량이 자동차 5대가 구매 시점부터 폐차에 이르기까지 방출하는 탄소 소비량과 같을 정도로 많음을 지적함

<표 1-1> 모델의 정확도와 학습시간 증가
[Han2018a]

| | Error Rate | Train Time |
|-----------|------------|------------|
| ResNet18 | 10.76% | 2.5 days |
| ResNet50 | 7.02% | 5 days |
| ResNet101 | 6.21% | 1 week |
| ResNet152 | 6.16% | 1.5 week |

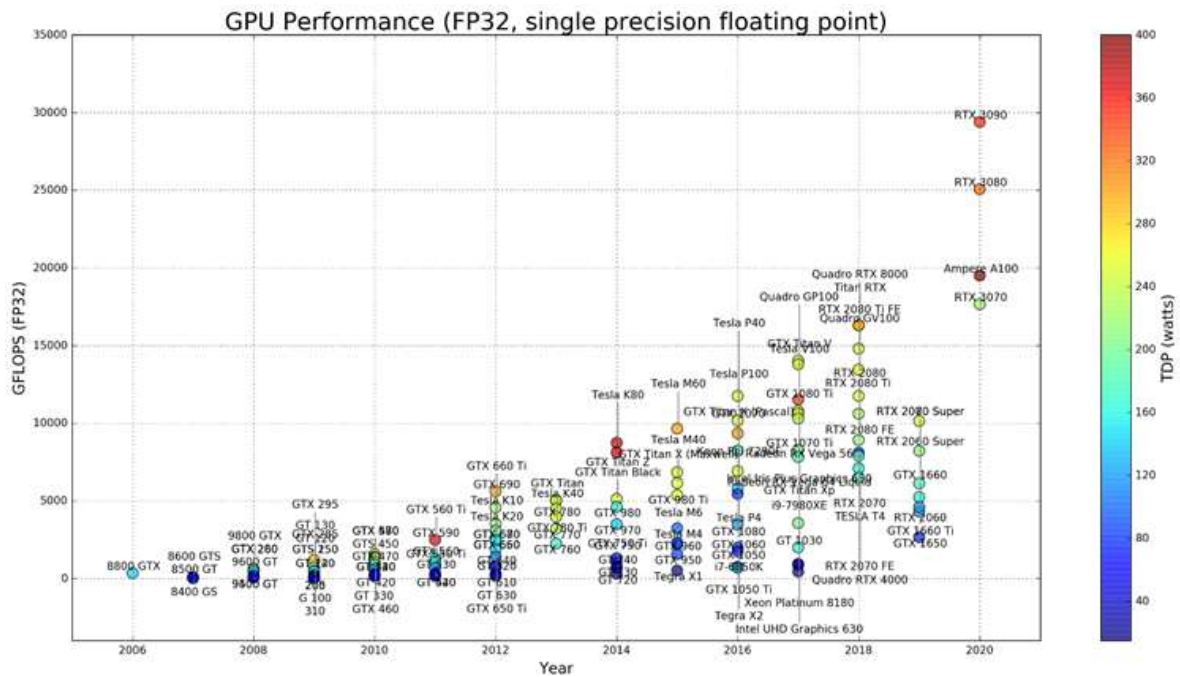
□ H/W 발전의 제한

- GPU는 그간 많은 발전을 이루어왔음에도 불구하고 딥 러닝모델의 크기와 연산량의 증가 추이와 비교하면 상대적으로 처리 연산량의 개선이 더딘 상태임
- 2012년 Tesla K20이 약 4,000 GFLOPS(FP32)의 연산량을 보였을 때 대비하

1) M40 GPU, fb.resnet.torch로 학습한 결과임

여, 2018년에 출시한 Titan RTX는 약 16,000 GFLOPS(FP32)로 6년간 약 4 배의 개선을 보였을 뿐임 [Sapunov2018a]

- 같은 기간 동안 GPU의 메모리는 약 3배 정도만 늘어났음
- 이는 앞서 언급한 바와 같이 모델 추론을 위한 연산량이 같은 기간 25배 증가한 것과 대비하여 6배 이상 개선이 더딤을 확인할 수 있음



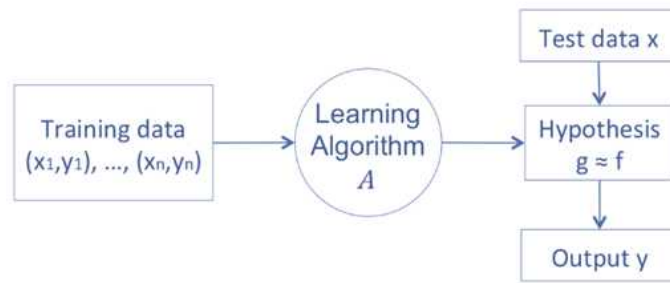
Quantization 기반 경량화 기술

제 1 절 기술 개요

1. 딥 러닝에서의 학습 과정 개요

□ 기계 학습 과정

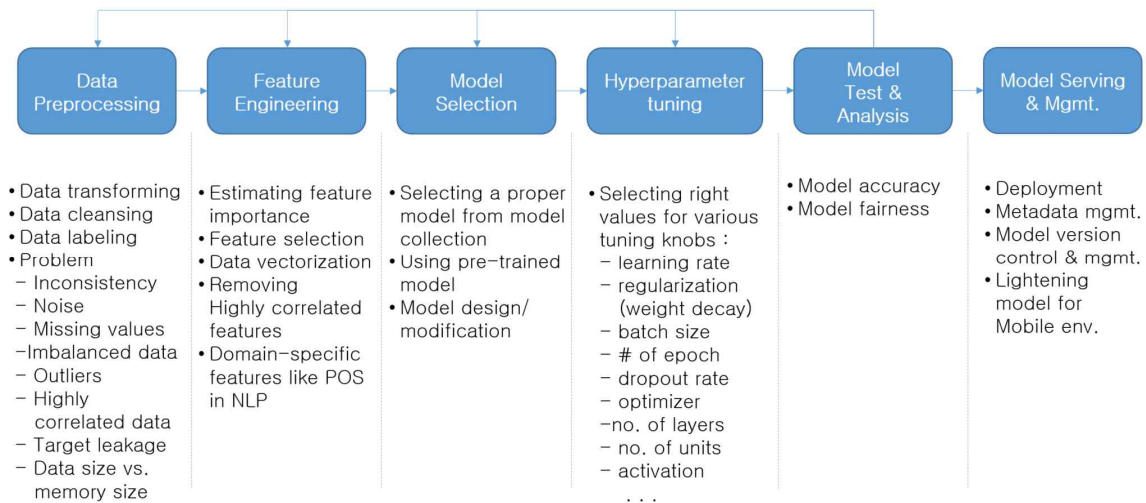
- 딥 러닝을 포함한 지도 학습(supervised learning) 방식의 기계학습 과정의 개념은 기본적으로는 아래에서 보이는 바와 같음



[그림 2-1] Overview of Supervised Machine Learning
[Polyzotis2017a]

- 먼저 $(x_1, y_1), \dots, (x_n, y_n)$ 쌍들로 구성되는 학습 데이터(labeled data)를 가짐
 - 이때 x 는 여러 개의 자질(feature)들로 구성되는 다차원 벡터이며, y 는 해당 x 에 대한 옳은 값임
 - 예를 들어 이미지 분류 문제의 경우 x 는 픽셀들의 조합으로 이루어진 이미지 데이터라 하면, y 는 해당 이미지를 설명하는 레이블 값으로 입력과 마찬가지로 원핫 벡터(one-hot vector) 등으로 표현됨
- 지도 학습은 이 학습 데이터를 가지고 학습 알고리즘(A)을 통해 1- 아직 알려지지 않은 대상 함수(unknown target function) g 에 대해 최대한 근사한 가설 함수(hypothesis) f 를 찾는 학습 과정과 2) 테스트 데이터를 이용하여 이 가설 함수가 기반 진실과 얼마나 일치하는지를 확인하는 테스트 과정으로 구분됨

- 하지만 딥 러닝 모델을 가지고 실세계에서 활용 가능한 응용(application)을 개발, 배포하기 위해서는 이보다 훨씬 더 복잡한 단계들을 거쳐야 함
 - 이때의 기계 학습 과정은 [그림 2-2]에서와 같이 여러 단계를 거치게 되며, 각 단계에서 개발자가 결정, 수행해야 할 작업 또한 다양함[Polyzotis2017a]
 - 이에 따라 딥 러닝 과정은 매 단계마다 사람의 손을 타야 하는 Human-In-The-Loop(이하 HITL) 프로세스로 정의할 수 있음[Xin2019a].
 - 또한 모델의 정확도가 개발자가 희망한 상태에 도달할 때까지 매 단계마다 조정을 거치면서 반복적으로 수행하는 특징이 있음
 - 더불어 시스템 입장에서 보자면, 학습 시간에 많은 시간이 소요됨에 따라 모델 학습에 있어 GPU 등 가용한 컴퓨팅 자원 모두를 활용하거나 파라미터 서버 방식[Li2014a, Abadi2016a] 등의 분산 학습을 통해 배치(batch) 작업으로 수행하여 학습 시간을 단축하고자 하는 특징이 있음
 - 이에 따라 학습 과정에서의 높은 컴퓨팅 자원의 점유율 및 전력 소비가 문제로 함께 제기되기도 한다. 예를 들면, 176 개의 GPU를 사용한 AlphaGo Fan의 경우 TDP(Thermal Design Power) 기준 40 kW이 넘었고, 약 3주의 학습 기간을 가짐[Silver2017a]
 - 따라서 효율적인 기계학습을 위해서는 모델의 정확도뿐만 아니라 모델 학습에 소요되는 각 단계별 시행착오를 줄여 HITL 프로세스 전체를 단축하는 것이 딥 러닝 모델 개발 과정에서의 자원 효율성 및 개발 시간을 단축하는데 있어 중요함
- 데이터 전처리 단계
 - 딥러닝 과정에서의 첫 단계는 데이터 전처리(data preprocessing) 단계로서 원시 데이터를 가공하여 학습을 위해 (x_i, y_i) 쌍으로 구성되는 벡터 또는 텐서 데이터로 변환하는 과정임
 - 이를 위해서는 기존 원시 데이터를 (x_i, y_i) 벡터 형식으로 변환하기 위한 데이터 변환, 데이터 간 불일치 및 노이즈 등을 해결하기 위한 데이터 클리닝,



[그림 2-2] Human-In-The-Loop process in Deep Learning

그리고 학습 데이터의 y 값에 해당하는 정보를 기입하는 레이블링 과정이 요구됨

- 이 과정에서 특히 중요한 작업들은 데이터 클린징(cleansing)과 레이블링(labeling)이라 불리는 작업들임
- 실세계에서의 원시 데이터는 [그림 2-2]에서 보이는 바와 같이 데이터의 불일치, 노이즈, 빠진 값들, 클래스 별로 상이한 데이터 량과 분포 등 다양한 문제를 가지고 있음에 따라 단순한 데이터 변환을 통해서만은 효과적인 학습 데이터를 갖추기 어려움
- 따라서 데이터 클린징 및 개체 해소(entity resolution) 기술[Doan2018a]을 통해 데이터를 정제하거나 추가적으로 관련 데이터를 보충하여 feature를 늘리는 등의 과정이 뒤따르게 됨
- 데이터 레이블링은 모델 학습을 위해 반드시 요구되는 작업임
 - 레이블링을 위한 여러 방법들은 이에 관한 가장 최근의 조사 논문인 [Roh2018a]를 참조하기 바람
 - 현재까지도 레이블링은 주로 사람이 직접 수행할 수밖에 없음에 따라 크라우드소싱(crowdsourcing) 방법에 크게 의존하고 있으며, 사람이 레이블링한 결과에 대한 검증 또한 중요 이슈로 부각되고 있음
 - 기계학습/AI 분야에서는 데이터 레이블링 문제를 해결하기 위해 전이학습

(transfer learning)이나 few-shot learning 과 같은 기술들이 개발되고 있음. 특히 언어 모델(language model)과 같은 경우 범용성을 가지고, 대량의 문서를 대상으로 미리 학습된 모델로, 이후 세부 작업에 맞는 모델 개발에 있어 이 언어 모델을 세부적으로 튜닝하는(fine-tuning) 과정을 거치는데 이것이 전이학습의 한 형태로 생각할 수 있음.

- 또한 데이터 프로그래밍 기법을 응용하여, 약한 지도 기반의 레이블링 생성 방법[Ratner2017a]도 존재하나 분류 문제에만 적용 가능한 한계를 가짐 [Roh2018a]

○ 자질 엔지니어링

- 두 번째 과정은 자질 엔지니어링(feature engineering)으로 전 처리된 입력 데이터를 벡터화(vectorization)하는 방법을 결정하거나, 중요 자질을 선택하는 과정, 그리고, 응용분야에 특화된 자질을 추가하는 작업 등을 수행
- 딥 러닝은 기존 기계학습 모델들과는 달리 자질 엔지니어링이 성능에 상대적으로 덜 영향을 미친다고 알려져 있음. 왜냐하면, 어떠한 자질들을 입력에 활용한다 하더라도, 다른 기계학습 모델들보다 상대적으로 모델 복잡도가 높아 중요 자질에 대한 활용을 보다 잘하는 것으로 알려져 있음
- 그러나 분야별로 전문지식이 요구되는 자질이나 자질간 연관성 등에 따른 불필요한 자질이 존재할 수 있으므로 이들에 대한 적절한 처리 또한 필요함
- 입력 데이터의 벡터화는 주로 원핫 인코딩(one-hot encoding), 자질 해싱(feature hashing), 레이블 인코딩 및 버킷타이징(bucketizing) 등의 다양한 벡터로의 인코딩을 통해 수행함
- 여러 자질 중 중요 자질의 선택을 위해서 중요도(importance)에 따라 자질을 선택적으로 이용하기도 하며, Part-Of-Speech (POS) 태그처럼 NLP 등과 같은 특정 분야에서 데이터 처리를 통해 활용 가능한 자질을 추가하기도 함
- 중요도에 따른 자질의 선택은 주로 응용분야 전문가에 의해 결정되었으나, 최근에는 SHAP[Lundberg2017a]이나 Model Class Reliance[Fisher2018a]와 같은 알고리즘을 이용하여 높은 중요도의 자질을 찾기도 함

○ 모델 선택 과정

- 세 번째 단계는 모델 선택 과정으로 응용에 적합한 모델을 선택하는 과정으로 여러 모델 중 적당한 모델 구조를 선택하여 학습을 진행하거나 또는 기존에 미리 학습된 모델(pre-trained model) 을 가져와 전이학습 방식으로 모델 학습을 수행하는 등의 과정임
- 하이퍼파라미터(hyper-parameter)는 모델 학습 과정을 시작하기 이전에 결정하는 파라미터로 크게 모델의 형태와 관계된 파라미터들과 모델 학습과 관계된 파라미터로 구분할 수 있음
 - 모델의 계층, 유닛 수, 활성화 함수(activation function) 유형 등은 모델의 형태와 관련된 하이퍼파라미터임.
 - 학습비율(learning rate)이나 미니배치 크기, epoch 수, 최적화 알고리즘의 선택 등은 모델 학습에 영향을 주는 파라미터임
- 모델의 다양성과 복잡성으로 인해 이러한 하이퍼파라미터의 설정은 최근까지 개발자의 반복적인 시행착오를 통해 점진적으로 최적화된 값으로 개선되고 있음
 - 하이퍼파라미터의 설정은 현재 많은 경우에 있어 개발자의 경험과 지식에 의존하여 설정하는 경향이 있음
 - 최근에는 AutoML 이라는 이름으로 모델 선택과 하이퍼파라미터 설정을 자동화하려는 연구 시도들도 존재함
 -

○ 모델 테스트 및 분석 과정

- 모델 테스트 및 분석 과정에서는 학습된 모델의 성능을 측정하여 분석하고, 이의 결과에 기초하여 모델 성능을 개선하기 위해 다시 HITL 프로세스의 이전 단계들에서 어떠한 작업을 수행할 것인지를 결정함
- 예를 들어 클래스별로 상이한 데이터 량이 모델 성능에 영향을 미친다 판단되면 SMOTE 기법[Chawla2002] 등을 이용하여 클래스 불균형을 해소할 수 있으며, 모델의 깊이를 키우거나 또는 모델 학습 관련 하이퍼파라미터 조정 등 다양한 결정사항들을 분석 결과에 기초하여 조정하고 모델 학습을 다시

수행

- 이 과정은 모델 학습의 테스트 결과가 개발자가 목표로 하는 성능이 나올 때까지 반복함. 따라서 모델 학습 과정의 시행착오를 줄이고 전체 학습 시간을 단축시키기 위해서는 유효한 컴퓨팅 자원뿐만 아니라 개발자의 경험이나 역량 또한 중요함
- 때문에 이러한 HITL 프로세스에서의 의사 결정사항들을 자동으로 결정할 수 있다면 딥러닝 과정에서의 HITL 프로세스에 요구되는 시간과 컴퓨팅 자원을 절약하는데 도움이 됨
- 이러한 모델 학습 과정의 자동화를 위한 노력으로 기계학습 분야에서는 AutoML (autonomous machine learning)으로 통칭되는 자동화된 학습 과정 지원에 대한 노력을 기울이고 있음 [Guyon2019a, He2018a]

○ 모델 서빙 및 배포

- 학습 완료된 모델은 실제 응용 분야에 적용되는데 이때는 모델의 동작 환경에 맞추어 모델을 경량화[Kang2019a]하기도 하며, 지속적인 서비스를 위해 모델 버전 관리를 수행
- 모델의 버전 관리는 특히 시간이 경과함에 따라 처리하는 데이터가 학습 데이터와 상이해지는 경우 재학습을 통해 모델의 정확도를 지속적으로 유지해야 함에 따라 요구됨. 이러한 버전 관리 및 학습에 사용한 데이터셋, 하이퍼파라미터 등의 정보 또한 관련한 메타데이터로서 관리[Miao2017a]가 필요함

2. CNN 모델에서의 학습 과정 개요

□ 이미지 처리와 필터링










- 이미지 필터링은 이미지 처리 분야에서 광범위하게 이용되고 있는 기법으로써, 이미지에서 테두리 부분을 추출하거나 이미지의 색상을 바꾸는 등 이미지를 구성하는 픽셀들을 모두 그대로 읽은 후 이에 대한 적절한 처리를 수행한 후 출력하는 기능을 의미함
- 이미지 필터링은 행렬 형태(또는 3차원 행렬로서 텐서라고도 함)로 표현된 이미지에 대해 마찬가지로 행렬로 표현된 커널(kernel; 필터라고도 함)을 동일하

게 적용하는 방식으로 수행됨

- 일반적으로 행렬로 표현되는 필터링된 이미지의 i 번째 행, j 번째 열의 픽셀인 G_{ij} 는 다음과 같이 원본 이미지 X 와 커널 F 간의 합성곱 (convolution) 연산으로 얻을 수 있음:

$$G_{ij} = (F * X)(i, j) = \sum_{m=0}^{F_H-1} \sum_{n=0}^{F_W-1} F_{m,n} X_{(i-m), (j-n)} \quad (\text{식 2-1})$$

- 여기에서 F_H 와 F_W 는 커널의 높이(행렬에서 행의 갯수)와 너비(행렬에서 열의 개수)임. 아래 그림은 다양한 종류의 필터와 각 필터의 기능을 보이고 있음

| Operation | Kernel ω | Image result $g(x,y)$ |
|--|---|--|
| Identity | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |  |
| Edge detection | $\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$ |  |
| | $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ |  |
| | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ |  |
| Sharpen | $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ |  |
| Box blur (normalized) | $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ |  |
| Gaussian blur 3×3 (approximation) | $\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ |  |
| Gaussian blur 5×5 (approximation) | $\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$ |  |
| Unsharp masking 5×5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask) | $\frac{-1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$ |  |

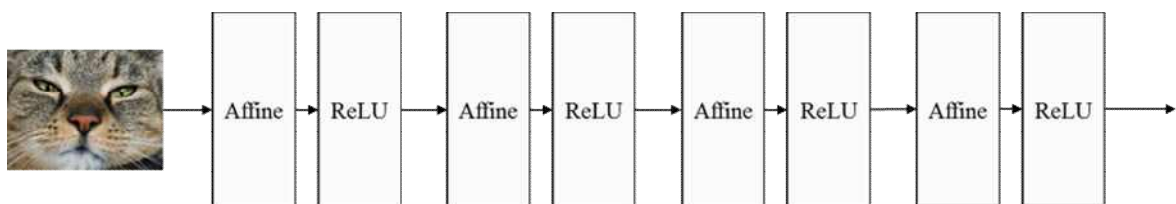
[그림 2-3] 커널의 종류와 동작 결과 예시
([en.wikipedia.org/wiki\(Kernel_\(image_processing\)\)](https://en.wikipedia.org/wiki/Kernel_(image_processing)))

□ 합성곱 신경망 (Convolutional Neural Network, CNN)

- CNN은 필터링 기법을 인공신경망에 적용함으로써 이미지를 더욱 효과적으로 처리하기 위해 LeCun 등에 의해 처음 소개[LeCun1989]되었고, 이후 LeCun 등에 의해 현재 딥 러닝에서 이용되고 있는 형태의 CNN이 학계에 보고됨 [LeCun1998]
- CNN의 기본 개념은 행렬로 표현된 필터의 각 요소가 데이터 처리에 적합하도록 자동으로 학습되게 하는 것임
 - 예를 들어, 이미지 분류 알고리즘을 개발하고자 할 때 필터링 기법을 이용하여 분류 정확도를 보다 향상시킬 수 있을 것임
- 그러나 여기에서 문제점은 어떻게 사용할 필터를 결정하는지에 관한 것임. 즉 기존에는 사람의 직관이나 반복적인 실험을 통해 알고리즘에 이용될 필터를 결정해야 했음. CNN은 이러한 필터를 선택하는데 있어 분류 정확도를 최대화하는 필터를 자동으로 학습할 수 있도록 함.

□ CNN의 구조

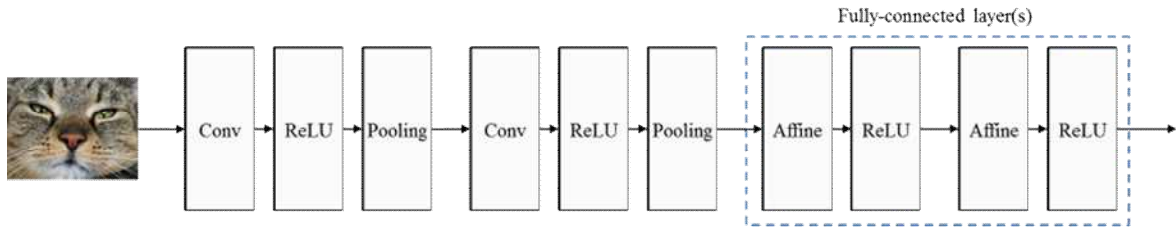
- 일반적인 뉴럴 네트워크의 구조는 그림 2-4에서 보이는 바와 같이 affine으로 명시된 완전 연결 계층(fully-connected layer)과 ReLU와 같은 비선형 활성화 함수 (nonlinear activation function)의 조합으로 여러 계층들을 쌓아 연결한 구조임



[그림 2-4] 일반적인 뉴럴 네트워크 구조

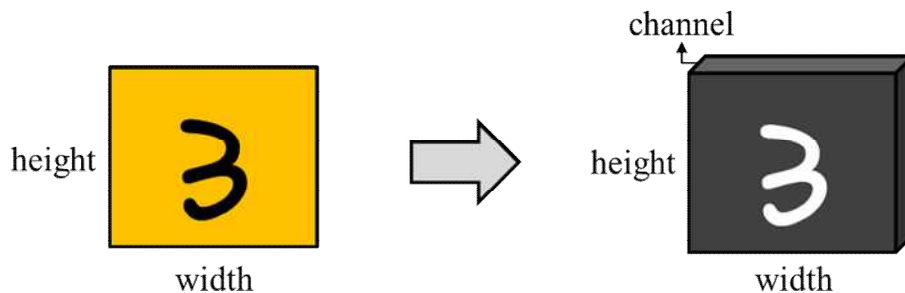
- CNN은 일반적인 뉴럴 네트워크의 구조를 그대로 따르나, 그림 2-5와 같이 합성곱 계층 (convolutional layer)과 풀링 계층 (pooling layer)이라고 하는 새로운 층을 완전 연결 계층 앞단에 추가함. 이는 원본 이미지에 대하여 커널을 이

용하여 필터링을 수행하고 그 뒤 필터링된 결과에 대하여 분류 연산이 수행되도록 모델을 학습하기 위함임



[그림 2-5] 일반적인 CNN의 구조 개요

- 합성곱 계층(그림 2-5에서 Conv로 명명된 계층)은 입력 이미지에 대해 커널을 적용하여 일종의 필터링 연산을 수행하고, 풀링 계층은 계산된 행렬의 값들을 하나의 대표적인 스칼라 값(행렬의 각 요소값들 중 최대값 또는 그 평균 등)으로 변환함으로써 이미지의 크기를 줄이는 등의 다양한 기능들을 수행
- 합성곱 계층(Convolutional Layer)
 - 이미지 데이터는 그림 2-6와 같이 높이×너비×채널의 3차원 행렬(또는 텐서(tensor)라고 명명함)으로 표현될 수 있음. 이미지의 색상은 주로 RGB 코드로 표현되는데, 이때 2D 이미지를 표현하는 2차원 행렬에서 한 픽셀의 값은 RGB 코드이므로 채널의 크기는 3이 되며 각 채널은 R, G, B 값을 하나씩 대표함

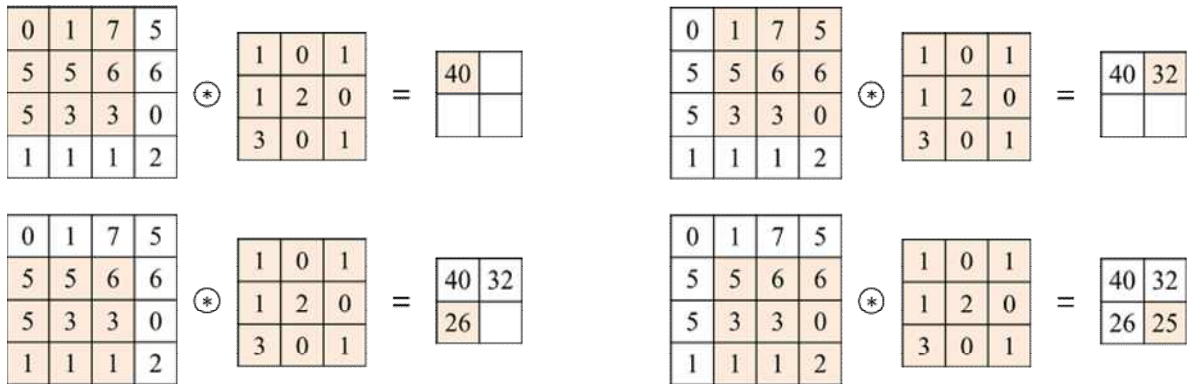


[그림 2-6] RGB 컬러 이미지 데이터에 대한 텐서 표현

- 하나의 합성곱 계층에는 입력되는 이미지의 채널 개수만큼 커널이 존재하며, 각 채널에 할당된 커널을 적용함으로써 합성곱 계층의 출력을 생성하게 됨
 - 예를 들어, 높이×너비×채널이 4×4×1인 텐서 형태의 입력 이미지에 대해

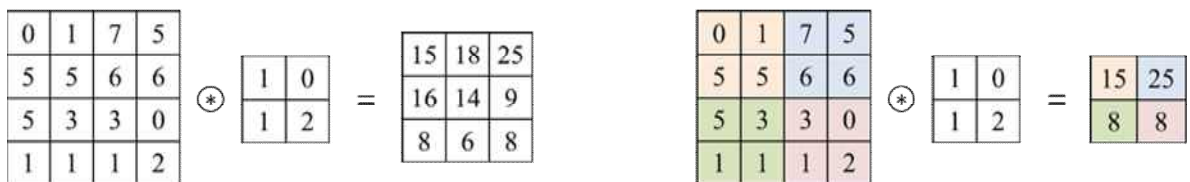
3x3 크기의 필터를 적용하는 합성곱 계층에서는 [그림 2-7]와 같이 이미지와 필터에 대한 합성곱 연산을 통해 2x2x1 텐서 형태의 이미지를 생성함.

- 그림에서는 커널의 범위만큼 입력 이미지의 픽셀값 하나하나를 커널에 대응하는 값과 곱하고, 이들을 모두 합한 값을 계산하여 출력함
- [그림 2-7]에서는 설명을 보다 쉽게 하기 위해 bias를 더하는 것이 생략되었는데, 실제 구현에서는 합성곱을 통해 생성된 행렬 형태의 이미지에 bias 값을 동일하게 더하도록 구현하기도 함



[그림 2-7] 하나의 채널에 대한 합성곱 계층의 동작.

- 이미지에 대해 커널을 적용할 때는 커널의 이동량을 결정하기 위해 스트라이드(stride) 값을 설정할 수 있음. [그림 2-8]은 동일한 입력 이미지와 커널에 대해 스트라이드 값에 따른 출력의 차이를 보임



[그림 2-8] 스트라이드가 1로 설정된 경우 (좌)와 2로 설정된 경우 (우)

- CNN을 구현할 때 합성곱 계층의 스트라이드는 주로 1로 설정되며, [그림 2-7]의 예제 또한 스트라이드가 1로 설정된 경우임

- [그림 2-9]에서 볼 수 있듯이 입력 이미지에 대해 합성곱을 수행하면, 출력 이미지의 크기는 입력 이미지의 크기보다 작아짐. 때문에 합성곱 계층을 거치면서 이미지의 크기는 계속 작아지게 되고, 이미지의 가장자리에 위치한 픽셀들의 정보는 점점 사라짐
- 이 문제를 해결하기 위해 패딩(padding) 기법을 적용하기도 함. 패딩은 아래 [그림 2-9]와 같이 입력 이미지의 가장자리에 픽셀들을 추가하여 입력 이미지와 출력 이미지의 크기를 같거나 비슷하게 만드는 역할을 수행함. [그림 2-9]의 예시처럼 이미지의 가장자리에 0의 값을 갖는 픽셀을 추가하는 것을 zero-padding이라고 하며, CNN에서는 주로 이를 이용함

Figure 2-9 illustrates the effect of zero-padding on convolution. The left part shows a 4x4 input matrix:

| | | | |
|---|---|---|---|
| 0 | 1 | 7 | 5 |
| 5 | 5 | 6 | 6 |
| 5 | 3 | 3 | 0 |
| 1 | 1 | 1 | 2 |

multiplied by a 3x3 kernel:

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 2 | 1 |
| 1 | 2 | 3 |

resulting in a 2x2 output:

| | |
|----|----|
| 41 | 33 |
| 25 | 23 |

The right part shows a 6x6 input matrix (with zero-padding):

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 7 | 5 | 0 |
| 0 | 5 | 5 | 6 | 6 | 0 |
| 0 | 5 | 3 | 3 | 0 | 0 |
| 0 | 1 | 1 | 1 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

multiplied by the same 3x3 kernel:

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 2 | 1 |
| 1 | 2 | 3 |

resulting in a 4x4 output:

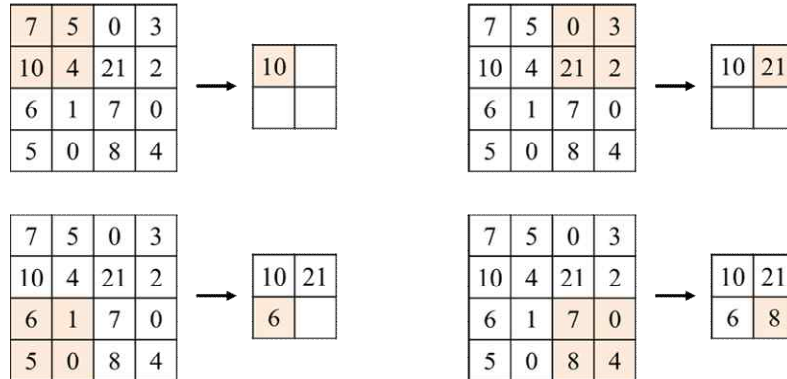
| | | | |
|----|----|----|----|
| 26 | 42 | 55 | 35 |
| 34 | 41 | 33 | 28 |
| 18 | 25 | 23 | 14 |
| 3 | 9 | 8 | 8 |

[그림 2-9] zero-padding을 적용하지 않은 경우(좌)와 적용한 경우의 합성곱 연산 (우).

□ 풀링 계층 (Pooling Layer)

- CNN에서 합성곱 계층과 ReLU와 같은 비선형 활성화 함수를 거쳐서 생성된 이미지는 풀링 계층에 입력하여 값의 개수를 축소함
 - CNN에서 풀링 계층으로는 주로 max-pooling을 이용함. 아래의 그림은 스트라이드가 2로 설정된 경우에 있어 max-pooling 기반 풀링 계층의 동작을 보임
 - [그림 2-10]의 예시에서는 2x2 크기의 국소 영역마다 max-pooling을 적용했으며, 일반적으로 풀링 계층의 스트라이드는 국소 영역의 높이 또는 너비의 크기와 동일하게 설정됨
 - 이미지 데이터의 특징은 인접 픽셀들 간의 유사도가 매우 높다는 것으로 이미지는 특정 속성을 갖는 국소 영역 수준으로 표현될 수 있으며, 풀링 계층은 이미지 데이터의 이러한 특징을 바탕으로 이용함. Max-pooling의 경우에는 국소 영역에서 가장 큰 값을 해당 영역의 대표 값으로 설정하는 것과 같은 효과임. CNN에서는 이러한 풀링 계층을 이용함으로써 여러 이

점을 가질 수 있음

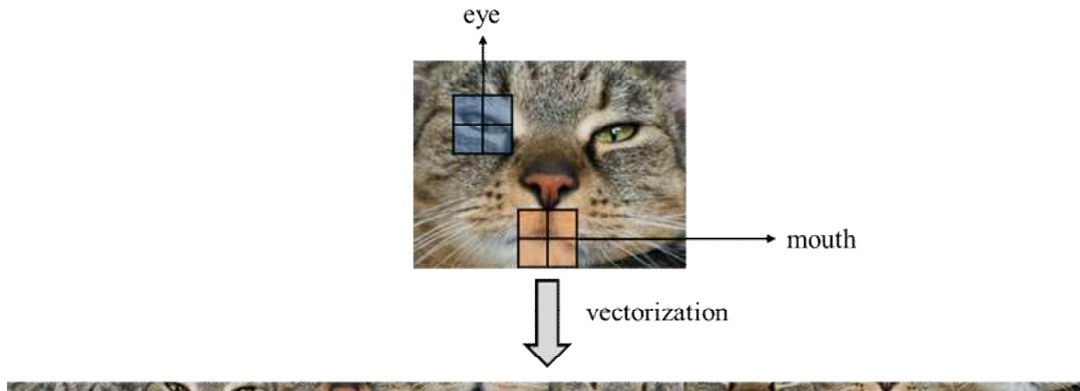


[그림 2-10] Max-pooling 기반 풀링 계층의 동작.

- 풀링 계층을 이용할 경우, 국소 영역에서 픽셀들이 이동 및 회전 등에 의해 위치가 변경되더라도 출력 값이 동일할 수 있게 되어 이미지의 이동 및 회전 등에 의해 CNN의 출력 값이 영향을 받는 문제가 완화됨
- 또한 처리해야 할 이미지의 크기가 줄어들게 되어 뉴럴 네트워크를 구성하는 파라미터 개수 또한 크게 줄어들어 학습 시간이 절약되고, 과적합 (overfitting) 문제 또한 완화되는 효과가 있음

□ 완전 연결 뉴럴 네트워크의 한계와 CNN의 장점

- 완전 연결 뉴럴 네트워크(FCNN; Fully-Connected neural network)은 인접 픽셀 간의 상관관계가 CNN과 대비하여 많이 무시됨
 - 완전 연결 뉴럴 네트워크는 벡터 형태로 표현된 데이터를 입력 받도록 고안되어 있기 때문에 [그림 2-11]와 같이 이미지를 입력 데이터로 사용하기 위해서는 벡터화를 수행하여야 함



[그림 2-11] FNN을 이용하여 이미지를 처리하기 위한 이미지 벡터화

- 그러나 이미지 데이터에서는 인접 픽셀간에 높은 상관관계가 존재하기 때문에 이미지를 벡터화(vectorization)하는 과정에서 인접 픽셀이 무엇인지 알기 어렵게 됨
- 반면 CNN은 행렬 형태의 데이터를 입력 받기 때문에 이미지의 형태를 보존됨. 이러한 이유로 이미지를 다루는 문제에 대해서는 대부분 CNN을 이용함

○ 모델 파라미터 수

- 만약 완전 연결 뉴럴 네트워크(FCNN)를 이용하여 $1,024 \times 1,024$ 크기의 컬러 이미지를 처리하고자 하면, FCNN에 입력되는 벡터의 차원은 약 315만 ($=1,024 \times 1,024 \times 3$)이 된다. 300만 차원의 입력을 처리하기 위해서는 필연적으로 막대한 양의 뉴런이 필요하고, 이에 따라 뉴럴 네트워크 전체의 파라미터 개수는 기하급수적으로 증가함
- 그러나 CNN을 이용한다면 컨볼루션을 통해 파라미터가 공유되는 효과가 있으며 또한 풀링 계층을 통과하면서 그 개수는 더욱 줄어들게 된다.
- 때문에 CNN은 기존 FCNN보다 매우 적은 파라미터를 가지며, 이미지뿐만 아니라 다른 고차원 데이터의 처리에 있어서도 널리 활용됨

□ 역전파 (Back propagation) 알고리즘을 이용한 CNN 학습

- CNN 또한 역전파(backpropagation) 알고리즘을 이용하여 모델 학습을 진행함

- 스트라이드가 1로 설정되었다고 가정할 경우, CNN의 l 번째 합성곱 계층의 출력 이미지 $X^{(l)}$ 의 k 번째 채널에 대한 i 번째 행, j 번째 열의 요소는 다음과 같이 계산됨

$$X_{k,i,j}^{(l)} = \sum_{m=0}^{F_H^{(l)}-1} \sum_{n=0}^{F_W^{(l)}-1} W_{k,m,n}^{(l)} f(X_{k,(i+m),(j+n)}^{(l-1)}) + b_k^{(l)}. \quad (\text{식 2-2})$$

- 위의 식 (2-2)에서 $w_k^{(l)}$ 은 l 번째 합성곱 계층의 k 번째 커널, $F_H^{(l)}$ 과 $F_W^{(l)}$ 은 각각 1번째 합성곱 계층 필터의 높이와 너비를 의미하고, f 는 ReLU와 같은 비선형 활성화 함수(nonlinear activation function)를 의미함
- 식 (2-2)에서 정의된 합성곱 연산에서는 식 (2-1)에서 합성곱 연산을 정의한 것과 다르게 행렬로 표현된 이미지를 m 과 n 만큼 증가된 방향으로 읽어감. 즉, 합성곱 연산 시에 (i,j) 에 (m,n) 를 빼는 것이 아니라 더해가면서 커널을 적용함
- 이는 CNN에서 구현의 효율성 문제에 기인한 것으로 합성곱 연산을 상호 상관(cross-correlation) 연산으로 구현함에 따른 것으로 합성곱 연산과 상호 상관 연산은 적용되는 커널을 회전시켜 적용한다는 점에서 차이가 있음
- 뉴럴 네트워크에서 가중치 행렬 W 의 학습을 위해 사용하는 경사하강법 (gradient descent)에 기반을 둔 역전파(backpropagation) 알고리즘은 아래와 같이 계산됨

$$W = W - \eta \frac{\partial L}{\partial W} \quad (\text{식 2-3})$$

- 이 식에서 L 은 손실 함수(loss function), η 는 경사값을 얼마나 반영하여 가중치를 감쇄시킬지 결정하는 하이퍼파라미터인 학습률(learning rate)임
- CNN에서는 가중치는 합성곱 계층의 커널에 해당하기 때문에 CNN을 학습시킨다는 것은 커널의 각 요소를 학습하는 것과 같음. 이를 위해 CNN의 l 번째 합성곱 계층에서 k 번째 필터의 i, j 번째 요소인 $W_{k,i,j}^{(l)}$ 에 대해 손실 함수 L 을 편미분하면 아래와 같음

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{W}_{k,i,j}^{(l)}} &= \sum_{m=0}^{\mathbf{X}_H^{(l-1)} - F_H^{(l)}} \sum_{n=0}^{\mathbf{X}_W^{(l-1)} - F_W^{(l)}} \frac{\partial L}{\partial \mathbf{X}_{k,m,n}^{(l)}} \frac{\partial \mathbf{X}_{k,m,n}^{(l)}}{\partial \mathbf{W}_{k,i,j}^{(l)}} \\
&= \sum_{m=0}^{\mathbf{X}_H^{(l-1)} - F_H^{(l)}} \sum_{n=0}^{\mathbf{X}_W^{(l-1)} - F_W^{(l)}} \delta_{k,m,n}^{(l)} \frac{\partial \mathbf{X}_{k,m,n}^{(l)}}{\partial \mathbf{W}_{k,i,j}^{(l)}}.
\end{aligned} \tag{식 2-4}$$

- 위의 식 (2-4)에서 $\mathbf{X}_H^{(l-1)}$ 과 $\mathbf{X}_W^{(l-1)}$ 은 각각 $l-1$ 번째 합성곱 계층에서 출력된 이미지의 높이와 너비를 의미함. 식 (2-4)를 계산하기 위해 $\partial \mathbf{X}_{k,m,n}^{(l)} / \partial \mathbf{W}_{k,i,j}^{(l)}$ 을 다음과 같이 계산:

$$\frac{\partial \mathbf{X}_{k,m,n}^{(l)}}{\partial \mathbf{W}_{k,i,j}^{(l)}} = \frac{\partial}{\partial \mathbf{W}_{k,i,j}^{(l)}} \left\{ \sum_{p=0}^{F_H^{(l)}-1} \sum_{q=0}^{F_W^{(l)}-1} \mathbf{W}_{k,p,q}^{(l)} f\left(\mathbf{X}_{k,(m+p),(n+q)}^{(l-1)}\right) + b_k^{(l)} \right\}. \tag{식 2-5}$$

- 미분 과정에서 $\mathbf{W}_{k,i,j}^{(l)}$ 과 독립적인 항은 모두 0이 되므로 식 2-5에서 $p=i$ 그리고 $q=j$ 인 항만 고려하여 식을 쓰면 다음과 같음:

$$\frac{\partial \mathbf{X}_{k,m,n}^{(l)}}{\partial \mathbf{W}_{k,i,j}^{(l)}} = \frac{\partial}{\partial \mathbf{W}_{k,i,j}^{(l)}} \left\{ \mathbf{W}_{k,i,j}^{(l)} f\left(\mathbf{X}_{k,(i+m),(j+n)}^{(l-1)}\right) \right\}. \tag{식 2-6}$$

- 따라서, $\partial \mathbf{X}_{k,m,n}^{(l)} / \partial \mathbf{W}_{k,i,j}^{(l)}$ 은 아래의 식 2-7과 같이 계산:

$$\frac{\partial \mathbf{X}_{k,m,n}^{(l)}}{\partial \mathbf{W}_{k,i,j}^{(l)}} = f\left(\mathbf{X}_{k,(i+m),(j+n)}^{(l-1)}\right). \tag{식 2-7}$$

- 그 다음, $\delta_{k,m,n}^{(l)} = \partial L / \partial \mathbf{X}_{k,m,n}^{(l)}$ 은 다음과 같이 계산함

$$\begin{aligned}
 \delta_{k,m,n}^{(l)} &= \frac{\partial L}{\partial \mathbf{X}_{k,m,n}^{(l)}} \\
 &= \sum_{p=0}^{F_H^{(l+1)}-1} \sum_{q=0}^{F_W^{(l+1)}-1} \frac{\partial L}{\partial \mathbf{X}_{k,(m-p),(n-q)}^{(l+1)}} \frac{\partial \mathbf{X}_{k,(m-p),(n-q)}^{(l+1)}}{\partial \mathbf{X}_{k,m,n}^{(l)}} \\
 &= \sum_{p=0}^{F_H^{(l+1)}-1} \sum_{q=0}^{F_W^{(l+1)}-1} \delta_{k,(m-p),(n-q)}^{(l+1)} \frac{\partial \mathbf{X}_{k,(m-p),(n-q)}^{(l+1)}}{\partial \mathbf{X}_{k,m,n}^{(l)}}.
 \end{aligned} \tag{식 2-8}$$

- 식 (2-8)에서 $\partial \mathbf{X}_{k,(m-p),(n-q)}^{(l+1)} / \partial \mathbf{X}_{k,m,n}^{(l)}$ 은 다음과 같이 계산:

$$\frac{\partial \mathbf{X}_{k,(m-p),(n-q)}^{(l+1)}}{\partial \mathbf{X}_{k,m,n}^{(l)}} = \frac{\partial}{\partial \mathbf{X}_{k,m,n}^{(l)}} \left\{ \sum_{r=0}^{F_H^{(l+1)}-1} \sum_{s=0}^{F_W^{(l+1)}-1} \mathbf{W}_{k,r,s}^{(l+1)} f \left(\mathbf{X}_{k,(m-p+r),(n-q+s)}^{(l)} \right) \right\}. \tag{식 2-9}$$

- 미분 과정에서 $\mathbf{X}_{k,m,n}^{(l)}$ 과 독립적인 항은 모두 0이 되기 때문에 식 2-9에서 $m-p+r=m$ ($r=p$)과 $n-q+s=n$ ($s=q$)인 항만을 고려하여 식을 쓰면 다음과 같음:

$$\frac{\partial \mathbf{X}_{k,(m-p),(n-q)}^{(l+1)}}{\partial \mathbf{X}_{k,m,n}^{(l)}} = \frac{\partial}{\partial \mathbf{X}_{k,m,n}^{(l)}} \left\{ \mathbf{W}_{k,p,q}^{(l+1)} f \left(\mathbf{X}_{k,m,n}^{(l)} \right) \right\}. \tag{식 2-10}$$

- 따라서, $\partial \mathbf{X}_{k,(m-p),(n-q)}^{(l+1)} / \partial \mathbf{X}_{k,m,n}^{(l)}$ 은 다음과 같이 계산된다:

$$\frac{\partial \mathbf{X}_{k,(m-p),(n-q)}^{(l+1)}}{\partial \mathbf{X}_{k,m,n}^{(l)}} = \mathbf{W}_{k,p,q}^{(l+1)} f' \left(\mathbf{X}_{k,m,n}^{(l)} \right). \tag{식 2-11}$$

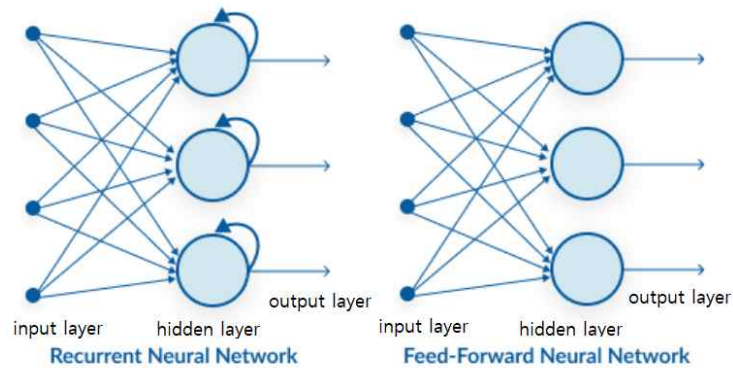
- 최종적으로, $\delta_{k,m,n}^{(l)}$ 은 다음과 같다:

$$\delta_{k,m,n}^{(l)} = \sum_{p=0}^{F_H^{(l+1)}-1} \sum_{q=0}^{F_W^{(l+1)}-1} \delta_{k,(m-p),(n-q)}^{(l+1)} \mathbf{W}_{k,p,q}^{(l+1)} f'(\mathbf{X}_{k,m,n}^{(l)}). \quad (\text{식 2-12})$$

- 식 2-7과 2-12의 결과를 식 2-4에 대입하면 $W_{k,i,j}^{(l)}$ 에 대한 L 의 기울기 (gradient)를 계산할 수 있으며, 이를 이용하여 식 2-3으로 $W_{k,i,j}^{(l)}$ 를 갱신함 Bias인 $b^{(l)}_k$ 또한 위와 같은 방법으로 갱신함

3. RNN 모델에서의 학습 과정 개요

- recurrent neural network은 convolutional neural network이 주로 이미지와 같은 grid값들을 처리하는 것과 달리, 시간 흐름에 따른 입력 데이터의 변화가 있는 sequence 데이터 처리에 유용한 특징을 지님. CNN 이 이미지의 입력 데이터의 상하좌우 크기변화에 유연한 것과 달리 RNN은 긴 Sequence처리에 유연한 장점을 지님. 이는 여러 time step에 걸쳐 같은 weights(가중치)를 공유하여 사용하는 특징에 기인함 ([그림 2-12] 참조) [Goodfellow, 2016]



[그림 2-12] Recurrent Neural Network과 Feed-Forward Neural Network 구조의 차이 (RNN에서는 hidden layer에서 여러 time step에 걸쳐 weight들이 공유되어 사용됨)

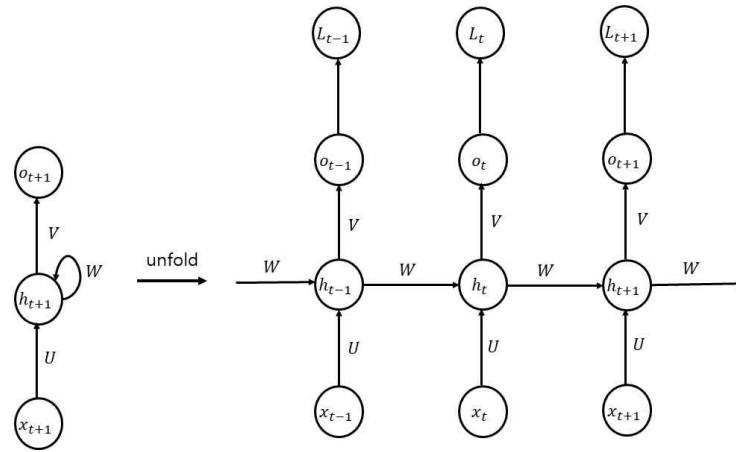
- graph unrolling 방법과 가중치 파라미터 공유 방법을 다양화하여 다양한 RNN을 구성할 수 있음.
- 기본적인 RNN의 Forward 연산 구조는 다음 수식과 [그림 2-13]과 같이 정

의 가능함. h_t 는 t time step에서의 hidden layer를, O_t 는 t time step에서의 출력을, L_t 는 t time step에서의 loss를, y_t 는 t time step에서의 목표 값, b , c 는 각각 bias를 나타내며, W_t 는 t time step에서의 가중치 값을 나타냄

$$h_t = f(Wh_{t-1} + Ux_t + b) \quad (\text{식 2-13})$$

$$O_t = c + Vh_t \quad (\text{식 2-14})$$

$$\hat{y}_t = \text{softmax}(O_t) \quad (\text{식 2-15})$$



[그림 2-13] recurrent neural network

보통 RNN에서는 사용되는 함수 f 로 표현된 활성화함수로 \tanh 가 사용됨.

RNN은 각 현재 시간의 Loss는 누적되어 쌓인 Loss로 가정됨. 현재시간 t 의 출력결과를 최종 시간을 τ 라고 가정하였을 때, 최종시간까지 영향을 미치게 되므로 $L_t = \sum_{j=t}^{\tau} L_j$ 이며, $\frac{\partial L}{\partial O_t} = \frac{\partial(L_t + L_{t+1} + \dots + L_{\tau})}{\partial O_t} = \frac{\partial L_t}{\partial O_t}$ 임.

- RNN의 Backward연산과정은 앞서 정의된 수식들을 기반으로 다음과 같이 정의됨.

시간 t 에서의 i th hidden node에 대한 출력 O_t 의 기울기 값은

$$\frac{\partial L_t}{\partial O_t^{i_t}} = (\nabla_{O_t} L)_i = \frac{\partial L}{\partial L_t} \frac{\partial L_t}{\partial O_t^i} = \hat{y}_t^i - 1_{y_t^i} \quad (\text{식 2-16})$$

이며, 현재시간 t 에 영향을 미치는 최종 시간 τ 에서부터의 Loss를 고려하여

hidden layer에 대한 loss의 영향은 다음과 같이 계산 가능함.

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L_t}{\partial \mathbf{h}_t} + \frac{\partial L_{t+1}}{\partial \mathbf{h}_t} + \frac{\partial L_{t+2}}{\partial \mathbf{h}_t} + \dots + \frac{\partial L_\tau}{\partial \mathbf{h}_t} \quad (\text{식 2-17})$$

마지막 시간 τ 에 대해 먼저 gradient를 계산하고,

$\frac{\partial \mathbf{o}_\tau}{\partial \mathbf{h}_\tau}$ 를 $\nabla_{\mathbf{o}_\tau} L$ 로 대체하여 표현하여,

$$\frac{\partial L_\tau}{\partial \mathbf{h}_\tau} = \frac{\partial L_\tau}{\partial \mathbf{o}_\tau} \frac{\partial \mathbf{o}_\tau}{\partial \mathbf{h}_\tau} = \mathbf{V}^T(\nabla_{\mathbf{o}_\tau} L) \quad (\text{식 2-18})$$

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L_t}{\partial \mathbf{h}_t} + \frac{\partial L_{t+1}}{\partial \mathbf{h}_t} = \frac{\partial L_t}{\partial \mathbf{O}_t} \frac{\partial \mathbf{O}_t}{\partial \mathbf{h}_t} + \frac{\partial L_{t+1}}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} \quad (\text{식 2-19})$$

위에서와 같은 방법으로 $\frac{\partial L_t}{\partial \mathbf{h}_t}$ 를 $\nabla_{\mathbf{h}_t} L$ 로 대체하여 표현하면,

$$\nabla_{\mathbf{h}_t} L = \mathbf{V}^T(\nabla_{\mathbf{o}_t} L) + \nabla_{\mathbf{h}_{t+1}} L \cdot \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} \quad (\text{식 2-20})$$

위 수식에서 $\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} = \frac{\partial \tanh(\mathbf{b} + \mathbf{W}\mathbf{h}_t + \mathbf{U}\mathbf{x}_t)}{\partial \mathbf{h}_t} = \mathbf{W}^T(1 - \mathbf{h}_{t+1}^2)$

$$\nabla_{\mathbf{h}_t} L = \mathbf{V}^T(\nabla_{\mathbf{o}_t} L) + \mathbf{W}^T(\nabla_{\mathbf{h}_{t+1}} L) \text{diag}(1 - \mathbf{h}_{t+1}^2)$$

$\text{diag}(1 - \mathbf{h}_{t+1}^2)$ 은 $(1 - \mathbf{h}_{t+1}^2)$ 의 대각행렬

back-propagation을 통해 구해야 하는 미분 값은

$$\frac{\partial L}{\partial \mathbf{U}'} \quad \frac{\partial L}{\partial \mathbf{W}'} \quad \frac{\partial L}{\partial \mathbf{V}'} \quad \frac{\partial L}{\partial \mathbf{b}'} \quad \frac{\partial L}{\partial \mathbf{c}}$$

\mathbf{W} 는 모든 간선에 기여하므로, 중의성 해소를 위해 시간단계 t 에서만 영향을 받는다고 가정하여 \mathbf{W}^t 를 도입하여 연산

$$\frac{\partial L}{\partial \mathbf{V}} = \frac{\partial L}{\partial \mathbf{O}_t} \frac{\partial \mathbf{O}_t}{\partial \mathbf{V}} = \sum_t^{\tau} (\nabla_{\mathbf{o}_t} L)(\mathbf{h}_t)^T \quad (\text{식 2-21})$$

같은 방법으로 각각 연산 하면,

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_t^{\tau} (\nabla_{\mathbf{h}_t} L) \cdot \text{diag}(1 - \mathbf{h}_{t+1}^2) \cdot (\mathbf{h}_{t-1})^T \quad (\text{식 2-22})$$

$$\frac{\partial L}{\partial \mathbf{U}} = \sum_t^{\tau} (\nabla_{\mathbf{h}_t} L) \cdot \text{diag}(1 - \mathbf{h}_{t+1}^2) \cdot (\mathbf{x}_t)^T \quad (\text{식 2-23})$$

$$\frac{\partial L}{\partial \mathbf{c}} = \sum_t^{\tau} (\nabla_{\mathbf{q}_t} L) \quad (\text{식 2-24})$$

$$\frac{\partial L}{\partial \mathbf{b}} = \sum_t^{\tau} \text{diag}(1 - \mathbf{h}_{t+1}^2) (\nabla_{\mathbf{h}_t} L) \quad (\text{식 2-25})$$

4. 모델 경량화 기술 개요

□ 개요

- 모델 경량화 기술은 모델의 정확도 손실(accuracy loss)을 기존 모델 대비 최소화하면서 모델 크기와 연산량을 크게 줄임으로써 요구 메모리와 에너지, 요구 연산량 등 여러 면에서 학습·추론의 효율성을 높이는 기술임
- 최근까지의 모델 경량화 기술들은 크게 1) 가지치기(pruning), 2) 양자화(quantization), 3) 지식 증류(knowledge distillation), 4) 경량 네트워크 설계(compact network design) 기법 4가지로 요약할 수 있으며, 각 기술에 대한 요약과 그 접근 방식은 <표 2-1>에서 보이는 바와 같음

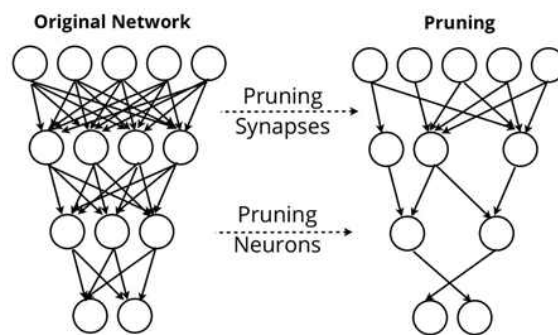
<표 2-1> 모델 경량화 기술의 분류

| 기술 분류 | 기술 요약 | 접근 방식 |
|--|---|-----------------------------|
| 가지치기 (Pruning) | 기준 값 이하의 가중치나 뉴런을 제거하며 반복 학습 | 기 학습된 모델을 이용 |
| 양자화 (Quantization) | 가중치, 활성화값, 기울기(구배)값을 보다 낮은 비트너비로 표현되는 값으로 대체 | 기 학습된 모델을 이용하거나, 모델 학습 때 적용 |
| 지식 증류 (Knowledge Distillation) | 기 학습된 모델을 teacher 모델로 하여 경량화된 student 모델을 학습함 | 기 학습된 모델을 이용 |
| 경량 네트워크 설계 (Compact Network Design) | 연산 효율성을 향상을 위한 뉴럴 네트워크 구조를 변경하는 방식 | 모델 학습 때부터 적용 |

- 이 중 가지치기와 지식 증류 기법은 기 학습된 모델(pre-trained model)을 먼저 준비하고 이를 기반으로 하여 별도의 경량화된 모델을 구하는 방법이며, 양자화 기법은 모델 학습 과정에서 경량화를 수행하거나 기 학습된 모델로부터 경량화 모두를 수행할 수 있는 기법임. 지식 증류 기법은 기 학습된 모델을 기반으로 경량화된 모델을 신규로 학습하는 방식이고, 경량 네트워크 설계 기법은 기존 비효율적인 모델 네트워크 구조를 개선하여 기존 모델 네트워크 대비 연산 량이나 파라미터 수를 줄이는 방식임

□ 가지치기 기법

- 가지치기 기법은 ‘모델의 모든 파라미터들이 추론에 동일한 영향을 미치는 것이 아니다’라는 관찰에서 출발하여, 추론에 있어 큰 영향을 미치지 못하는 뉴론들과 이들의 연결을 제거하는 방식으로 모델의 파라미터들을 줄여가며 반복 학습하여 파라미터들이 상당 부분 제거된 뉴럴 네트워크를 얻는 방법임 [Han2015a, Han2015b, Li2016a, Tung2018a]
- Han 등[Han2015a]은 특정 임계점을 설정하고, 이보다 낮은 값을 가지는 뉴론들과 이들의 연결들을 계층 단위로 제거하고, 재학습을 반복하는 방식으로 전체 파라미터 개수를 줄이는 방식을 제안하였음 ([그림 2-14] 참조). 그 결과 AlexNet[Alex2012a]을 기준으로 정확도 손실은 0.1%p(42.78% 대비 42.77%)에 불과했지만 파라미터 개수는 61M 개에서 6.7M로 약 9배를 축소하였음



[그림 2-14] 가지치기 기법 개요[17]

- Li 등[Li2016a]은 기 학습된 CNN 모델에서 덜 중요한 커널(필터)들을 이와 연결된 피쳐 맵(feature map)들과 함께 가지치기 하는 방법으로 계산량을 줄이는

방법을 제시하였음.

- 이 방법을 통해 이론적으로 m 개의 필터를 가지치기 하였을 때 각 계층 별로 (m /출력 채널수) 만큼의 계산량을 줄일 수 있음을 보였음
- [Han2015b]와 [Tung2018a]에서는 크게 기여하지 못하는 파라미터들을 가지치기(또는 clipping)하고, 이들을 클러스터링 한 후 해당 클러스터링의 중심값(centroid) 등으로 기존 파라미터들을 교체하는 일종의 양자화 기법 또한 같이 적용하였음. 이를 통해 [Tung2018a]의 경우 원래의 기 학습된 ResNet-50[He2016a, He2016b] 모델의 정확도는 오히려 0.6% 개선하면서도 102.5MB의 기존 모델 크기를 6.7MB로 약 15배 정도 더 경량화 시킬 수 있음을 보였음. [Han2015b]의 경우에는 VGG-16 모델을 최대 49배까지 경량화시켰음

- 그러나, 가지치기 기법은 기학습된 모델이 미리 준비되어 있어야 함. 즉 모델을 신규로 학습시킨 이후에 가지치기 기법을 적용함으로써 경량화를 수행함에 따라 학습 시작부터 최종적인 경량화된 모델을 얻기까지의 시간이 상대적으로 크게 소요됨. 또한 가지치기 기법은 반복적인 가지치기와 클러스터링 연산을 수행함에 따라 경량화에 소요되는 시간이 상대적으로 매우 높다는 제약을 가짐

□ 지식증류 기법

- 지식증류 기법(knowledge distillation 또는 그냥 distillation이라고도 함)은 기 학습된 여러 모델들을 가지고 모델 앙상블(ensemble)하여 이를 보다 작은 모델로 학습 시키기 위해 고안된 기법으로 teacher-student 모델로도 부름. 즉 기 학습된 모델들을 teacher 모델로 하고, 지식증류기법을 적용하여 얻은 작은 모델을 student 모델이라 명명함
- 지식 증류 기법은 2015년 Hinton[Hinton2015]에 의해 처음 제시된 이후 최근의 모델 경량화 연구에 많이 이용되고 있음. 초기 개발 의도는 모델 앙상블과 동일한 효과를 갖는 하나의 모델을 얻고자 하는 용도로 고안되었음. 현재는 이 지식 증류 기법을 활용하여 기 학습된 모델들을 임베디드 기기에서의 자원 제한을 고려하여 보다 적은 수의 학습 파라미터들과 적은 비트너비를 갖도록 student 모델을 구함으로써 모델이 저장될 저장 공간 및 실행 메모리 사용량을

줄일 수 있도록 함.

- 지식 증류 기법을 위해서는 먼저 기 학습된 모델이 teacher 모델로서 준비가 되어 있어야 한다.
- Polino 등[Polino2018a]은 모델 파라미터의 양자화와 함께 지식 증류를 통해 원 모델보다 훨씬 작은 모델을 생성할 수 있음을 보였고, FaceBook AI팀은 ICLR2020에서 발표한 kill-the-bits[Stock2020]를 통해 Song Han 교수팀의 가지치기와 양자화 기법이 결합된 HAQ 기법[Wang2019a]과 비등한 압축효율과 정확도를 보인록 지식증류 기법에 기반을 두고 최종 손실함수의 차이를 최소화 되도록 하는 프로덕트 기반 양자화(Product quantization) 기법을 함께 도입함
- Alibaba 연구팀은 CVPR 2019 학회에서 지식증류 기법을 기반으로 활성화 함수인 sigmoid 함수의 변형을 통해 양자화 함수 형태로 설계하고 완전정밀도 타입(full-precision) 기반의 학습 파라미터로부터 보다 낮은 비트너비(low bitwidth)를 갖도록 양자화된 파라미터들을 갖는 모델을 학습할 수 있는 방법을 제안함[Yang2019b].
- 이와외 별도로 지식 증류 기법은 원래 기존 모델을 기준으로 새로운 모델을 생성해 내기 위해 고안된 기법이지만 이 기법을 이용하여 새로운 학습 데이터를 합성하는 방식으로 최근에 이용되기도 한다[Wang2018a, Yin2020a].

제 2 절 Quantization 기법

1. 기술의 개요

- 양자화(Quantization) 기법은 FP32(32비트 부동소수점 타입) 기반의 학습 파라미터의 값들을 이보다 낮은 비트너비를 갖는 1~K 비트 크기로 표현하기 위해 사용함
- 양자화란 용어는 본래 아날로그 신호를 디지털로 변환하는데 있어 아날로그 값을 비트열로 표현할 수 있는 디지털 값으로 변환하는 과정을 일컫지만, 여기에서는 기존 부동소수점 타입으로 표현되던 값을 보다 적은 비트로 표현하기 위해 변환하는 과정을 의미함

- 많은 경우 딥 러닝 모델들은 파라미터 저장을 위해 FP32 타입을 주로 사용하는데, 이를 그보다 훨씬 낮은 K 비트들로 표현함으로써 가지치기 기법에서처럼 파라미터들의 일부를 버리지 않더라도 모델의 크기를 이론상 32/K 배 만큼 줄일 수 있는 장점을 가짐
- 반면에, 낮은 비트너비로 인해 표현할 수 있는 수의 범위가 크게 한정되는데 이는 그대로 정확도의 손실로 이어지는 문제가 발생함.
 - 예를 들어 32비트 부동소수점으로 표현되던 값을 1비트로 표현한다면, 0 또는 1과 같은 2개의 값으로 모든 값들을 대표해야함. 반면 값의 크기는 1/32 로 줄어들게 됨
- 때문에, 양자화를 수행하는데 있어 최종 목표는 보다 낮은 비트너비를 이용하면서도 FP32 타입 대비 정확도 손실을 최소화하는 것에 있음

| | | | |
|------|------|------|------|
| 1.12 | 3.42 | -1.5 | -12 |
| 32 | -1 | -5 | 15 |
| 24 | 0.55 | -54 | 0.24 |
| -0.1 | 0.1 | -0.2 | 2 |

→

| | | | |
|----|----|----|----|
| 1 | 1 | -1 | -1 |
| 1 | -1 | -1 | 1 |
| 1 | 1 | -1 | 1 |
| -1 | 1 | -1 | 1 |

[그림 2-15] 파라미터의 양자화(2진화)
예시

- 양자화 기법은 여러 경량화 기법과 함께 사용될 수 있는 특징 또한 가짐
 - [Han2015b, Tung2018a, Stock2020, Yang2019b] 등의 연구들이 이의 대표적인 경우임. 양자화 기법은 기 학습된 모델을 가지고 양자화를 수행하는 방법으로 이용할 수도 있지만, 학습과정 중에 양자화 함수를 통해 기 학습된 모델 없이 바로 경량화된 모델을 구할 수도 있음. 때문에 많은 경량화 기법들은 양자화 기법을 널리 활용하고 있음
- 양자화를 수행하는데 있어서 대상이 되는 값은 모델의 가중치(weight) 뿐만 아니라 활성화 출력(activation), 그리고 기울기(gradient; 또는 구배라고도 함) 모두가 해당이 될 수 있으며, 양자화 대상에 따른 이점과 문제점은 표 2-2에서 보이는 바와 같음

<표 2-2> 양자화 대상에 따른 분류 [Stock2020]

| Components | Benefits | Challenges |
|--------------------|--|--|
| Weight | <ul style="list-style-type: none"> • Smaller model size • Faster forward training & inference • Less energy | <ul style="list-style-type: none"> • Hard to converge with quantized weights • Require approximate gradients • Accuracy degradation |
| Activations | <ul style="list-style-type: none"> • Smaller memory foot print during training • Allows replacement of dot-products by bitwise operations • Less energy | <ul style="list-style-type: none"> • Gradient mismatch problem |
| Gradients | <ul style="list-style-type: none"> • Communication & memory savings | <ul style="list-style-type: none"> • Convergence requirement |

- 모델의 모든 파라미터들이 양자화의 대상이 되지만, 최근의 연구들을 보면 기울기의 경우에는 분포가 매우 극단적일 수 있고, 양자화 오류(quantization error)로 인해 학습 중 파라미터 값들이 converge가 잘 안 되는 문제로 인해 양자화 대상으로 크게 고려되고 있지는 않음
 - 물론 기울기 또한 양자화를 수행[Zhou2016a]하거나, 분산 학습에서의 기울기 값 전달에 있어서의 네트워크 I/O를 줄이고자 하는 시도[Jiang2018a] 또한 존재함
 - 하지만, 대다수의 연구들이 가중치와 활성화 출력을 양자화 대상으로 하므로 여기에서는 이러한 기법들을 위주로 설명하도록 함
- 양자화에 있어서 코드북(codebook)은 양자화를 수행한 이후의 파라미터 값을 지정함
- 이러한 코드북은 양자화 과정 전에 값이 지정이 되어 있는 경우에는 **고정 코드북(fixed codebook)**이라 하고, 양자화 과정 중에 양자화 값을 결정하는 코드북을 **적응형 코드북(adaptive codebook)**이라 함
 - 현재는 고정 코드북 보다는 적응형 코드북이 보다 낮은 정확도 손실(accuracy degradation)을 보임에 따라 적응형 코드북을 주로 사용하고 있음

2. 이진화 기법

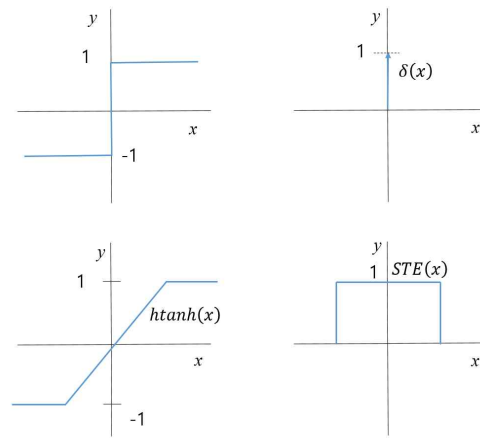
- 이진화(binanzation) 기법[Courbariaux2015a, Courbariaux2016a]은 코드북이 {-1, 1}로만 구성되는 양자화 기법으로 코드북이 가장 단순한 형태를 보임. 이진화

기법은 [그림 2-15]에서 보이는 바와 같이 가중치나 활성화 출력을 -1 또는 1로만 대표함

- 실제로 구현에서는 1 비트만으로 표현하는데 비트 값 0을 -1로 해석하는 방식으로 구동하므로 이론적으로 FP32 타입 모델 대비 32배(32비트→1비트)의 경량화가 가능함.
- Courbariaux 등[Courbariaux2015a]은 처음으로 모델의 가중치들에 대한 이진화 기법을 제안하였고, 그 후속연구인 [Courbariaux2016a]에서는 가중치와 활성화 출력 모두 이진화 하는 방법을 소개하였음 이를 위해서 결정론적 방법에서는 순전파(forward propagation) 시 가중치 값이 0 이상이면 +1로, 그 미만이면 -1로 이진화를 수행함.
- 이는 간단히 signum 함수로서 구현이 가능하므로 순전파시에는 signum(sign 함수라고도 함) 함수를 이용하여 구현하지만, 역전파(backward propagation) 시에는 signum 함수의 도함수(derivative)가 미분 가능하지 않음
- 때문에 역전파 시에는 signum 함수를 근사화하여 대체할 수 있도록 hard tanh(hyperbolic tangent) 함수의 도함수를 이용하며, 이 기법은 STE(Straight Through Estimator)[Bengio2013a] 이라고 하며 Bengio 등에 의해 제시함
- 순 전파에서의 함수와 역 전파에서의 도함수가 서로 일치하지 않는 관계로 기울기 값의 차이가 발생하는데, 이를 보완하기 위해 HAWQ[35]나 PACT[36]와 같은 STE에 대한 보완 연구 또한 존재함.
 - 이 두 연구는 모두 활성화 함수를 거친 값에 대하여 좀더 정확도 손실이 적도록 하면서 미분 가능한 값을 구하기 위해 ReLU 대신 사용하는 것으로 기본적으로는 ReLU와 유사하나 끝부분이 clipped 되는 형태를 취하며 어느 범위에서 clipped 될지는 학습에 의해 결정하도록 하고 있음 ([그림 2-16] 참조)
- 가중치와 활성화 출력을 모두 이진화하였을 때의 이점은 모델 크기의 축소뿐만 아니라 모델 학습·추론 과정에서 필수적인 행렬 곱셈 연산을 비트열 연산으로 대체할 수 있는 이점 또한 존재함
 - 이때의 비트열 연산은 XNOR 연산과 bitcount 연산으로 두 비트열에 대한

XNOR 연산 후 1의 개수를 계산하는 것으로 2진화된 두 비트벡터에 대한 행렬 곱셈 연산이 가능하다는 특징을 가짐. 이를 통해 연산 속도의 증가 또한 기대할 수 있음

- XNOR 연산과 bitcount 연산으로의 행렬 곱셈 연산의 대체는 BinaryNet[Courbariaux2016a]과 XNOR-Net[Rastegari2016a]에서도 보이고 있다. XNOR-Net의 경우에는 기본적으로 $\{-1, 1\}$ 코드북을 이용하지만 scale factor를 도입하여 좀 더 정확도 손실을 개선하고자 하였음



[그림 2-16] signum 함수와 STE

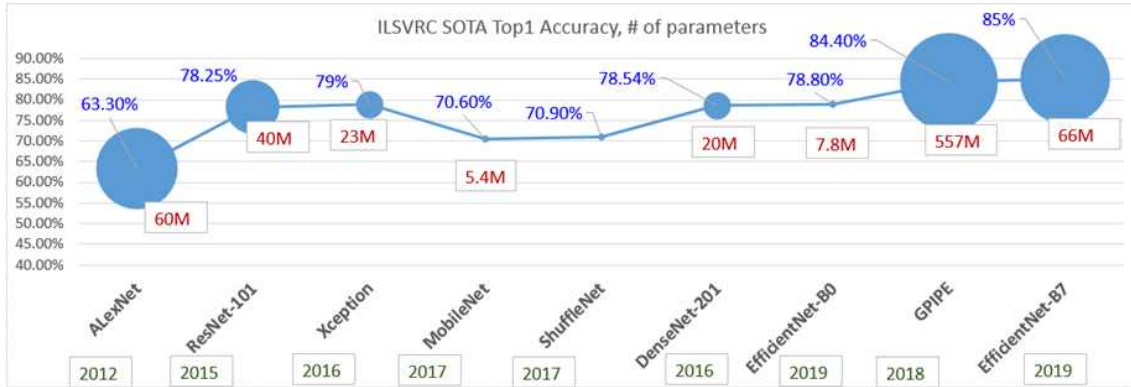
- 삼진화(ternarization) 기법[Alemdar2017a]은 이진화 기법으로 양자화를 수행하는데 있어 많은 가중치 값들이 0임을 확인하고, 코드북을 $\{-1, 0, 1\}$ 로 확장한 기법임

3. k-비트 양자화 기법

- k-비트 양자화 기법은 양자화에 있어 코드북을 임의의 k-비트너비로 표현할 수 있는 수만큼 필요에 따라 늘릴 수 있게 허용하는 특징이 있음. (예, $\{\pm v_1, \{\pm v_1 \pm v_2\}, \dots, \{\pm v_1, \dots, \pm v_k\}\}$).
- 특히 이 기법은 양자화 적용 대상에 따라 다른 k-비트너비 설정이 가능함(예, DoReFa-Net). 많은 경우 가중치보다 활성화 값에 보다 높은 k-비트너비를 설정

함

- DoReFa-Net[Zhou2016a]은 양자화의 대상을 가중치와 활성화 출력뿐만 아니라 기울기 값까지 양자화의 대상으로 함. 저자들의 실험에서는 가중치는 1, 활성화



[그림 2-17] 뉴럴 네트워크 구조에 따른 파라미터 수와 정확도의 변천

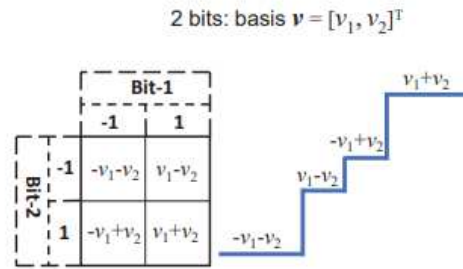
화 출력은 2, 기울기 값은 6비트로 하였을 때 가장 높은 정확도를 보였기에 이름 또한 이에 해당하는 음계의 이름을 따서 명명하였음

- LQ-Net[Zhang208a]은 k-비트너비의 지원 이외에 몇 가지 기법들을 추가로 소개하였다. 먼저 k-비트너비를 지원하면서도 비트열 연산자에 대한 호환성을 위해 벡터 분할 방식을 선보였고, 계층 별과 채널 별로 학습 가능한 양자화기 (learnable quantizer) 또한 제안하였다. 기존의 양자화 기법에서의 양자화기 (quantizer)들이 모두 고정된 코드북을 가지고 있었다면, LQ-Net은 모델 학습 과정 중에 양자화 오류(quantization error)가 최소가 되도록 양자화 구간을 모델의 계층 단위로 조정함(식 2-26 참조). 이는 각 계층의 가중치와 활성화 출력의 분포가 서로 상이하다는 관찰에 뿌리를 두고 있음

$$Q^*(x) = \arg \min_Q \int p(x) (Q(x) - x)^2 dx \quad (\text{식 2-26})$$

- 수식 (2-26)에서 $p(x)$ 는 x 의 pdf이고, $Q(x)$ 는 x 를 입력으로 하는 양자화 함수를 표현함. 또한 k-비트로 양자화를 함에 있어서도 비트열 연산이 가능 하도록 양자화 구간을 결정하는 기준 벡터(basis vector)와 실제 비트열 연산을 수행할 비트 벡터를 나눠 구분해 둠

- [그림 2-18]는 LQ-Net에서 양자화를 2비트로 수행할 경우 양자화 된 값을 결정하는 방식을 보이고 있음. 2비트로 4 개의 양자화 된 값을 설정할 수 있는데, 이때 각 비트가 표현하는 값은 기준 벡터 $\{v_1, v_2\}$ 을 참조하며, 식 2-26을 최적화시키기 위해 기준 벡터 값을 학습을 통해 보정함. 이렇게 할 경우 기준 벡터가 아닌 비트 벡터들 간에는 비트열 연산이 여전히 유효하다는 특징을 가짐



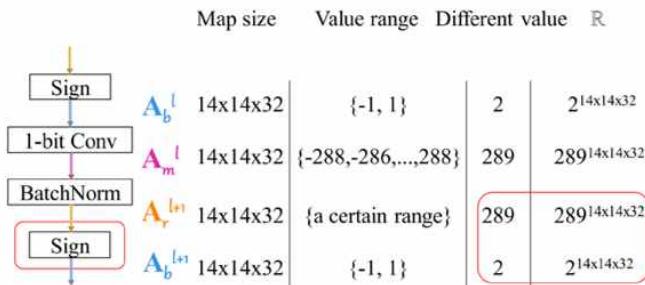
[그림 2-18] LQ-Net에서의 기준 벡터와 양자화 된 값의 예 ($k=2$ 인 경우)

- LQ-Net이 계층별 양자화 오류의 최적화를 목표로 함으로써 전체 모델의 정확도 손실을 최소화하고자 하는 반면에, 아예 전체 모델에서의 정확도 손실을 최소화하고자 하는 연구들도 존재하는데, kill-the-bits[Stock2020]와 같은 연구들은 전체 모델의 정확도 손실의 감소를 우선 고려함. 엄밀히 얘기하자면 DoReFa-Net이나 LQ-Net의 경우에는 기 학습된 모델로부터가 아닌 모델 학습 과정 중에 양자화를 함께 진행하는 모델이라는 점에서 손실 함수의 선택이 다르다 할 수 있음

4. Bi-Real Network

- 한 때 Binary Net의 State-Of-Art 모델이었던 Bi-Real net [Liu2018a]은 기존의 이진화 기법의 제약점을 개선하고자 3종류의 기술을 제안하였음
 - (1) 학습 파라미터의 표현 범위를 넓힘
 - (2) STE(Straight Through Estimator)에 사용된 Sign 함수 대신 polynomial 형태의 추정 Sign 함수를 적용하여 가중치의 미분 연산 시 실제 가중치 크기까지 고려함

- (3) 기 학습 모델을 기반으로 양자화하기 위한 초기화 함수를 non-negative RELU가 아닌 Clipped RELU를 적용함
- BiReal-Net에서도 양자화 계열 연구들이 정확도 성능을 높이고, 학습 시간을 줄이기 위해 사용해온 일반적인 방법들이 두 가지 적용됨. 정확도 성능을 높이기 위해 사용된 방법은 첫 번째 컨벌루션 계층과 마지막 완전 연결 계층(fully connected layer)에 양자화를 적용하지 않는 방법이고, 학습시간 단축을 위해서는 기학습 모델을 통해 학습된 파라미터를 시작으로 학습을 시작함. 여기에서는 특별히 학습 성능 향상을 위해 기학습된 모델을 학습 할 때 RELU가 아닌 Clipped RELU를 기반으로 학습시켜 정확도 성능도 향상시킴
- 학습 파라미터의 표현범위를 넓히는 방법으로 이진화후 동일 연결(identity connection) 대신 Batch normalization으로 그 연산 범위가 커진 상태에서 동일 연결을 두어 표현 범위를 증강 시키는 방법을 사용함.
- 보통 ResNet에서 두 개의 컨벌루션을 수행하고 Batch Normalization과 동일 연결을 수행함. 이를 하나의 컨벌루션과 Batch Normalization을 수행하고 이 때 마다 동일 연결시키는 방안을 제시함. 이를 통해 성능 향상에 이점이 있으나 연산량을 증가 시킨 제약점이 따름.



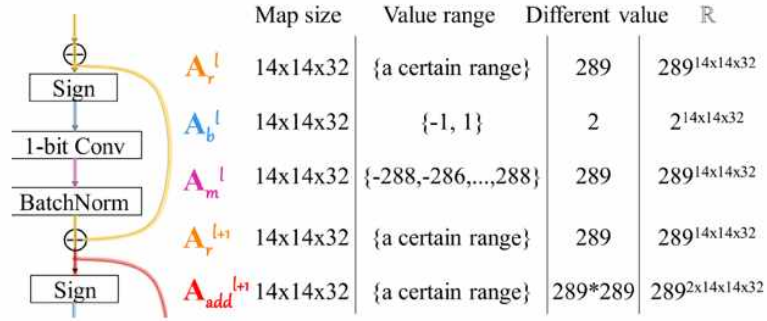
- Sign) 32channel with 14x14 feature map

$$R(A_b^l) = 2^{14 \times 14 \times 32} = 2^{6272}$$

[그림 2-19] $W \times H \times C$ feature map의 값의 표현 종류(D, different value)에

따른 표현 범위 $R(D, W \times H \times C) = D^{W \times H \times C}$

- STE에서 사용한 추정 Sign 함수는 부분 polynomial 함수를 적용한 것으로 역 전달(back propagation)시 가중치의 크기가 고려되어 적용되었다는 점에서 이전 STE방법과는 다른 형태임. 일반적인 STE에서 사용한 것과 마찬가지로 실제



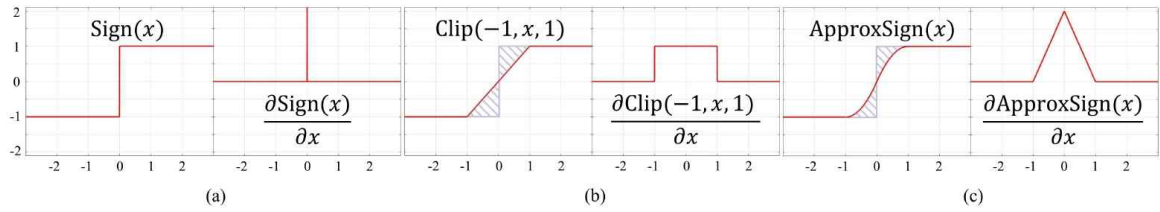
- After batchNorm,

$$R(A_r^{l+1}) = 289^{6272}$$

- By shortcut, the previous block input is added, thus

$$R(A_{add}^{l+1}) = (289^2)^{6272}$$

[그림 2-21] 학습 파라미터 표현 범위를 넓히기 위해 BatchNorm수행 후 바로 동일연결을 수행하도록 제안



[그림 2-20] (a) Sign함수와 그 미분 값, (b) Clip 함수와 그 미분 값, (c) Approximate Sign 함수와 그 미분값.

가중치의 부호 부분은 역 전달 연산 시 그 값을 그대로 취하는 방식을 취함. 추정 Sign 함수는 식 2-27와 [그림 2-21] 참조.

$$F(a_r) = \begin{cases} -1 & \text{if } a_r < -1 \\ 2a_r + a_r^2 & \text{if } -1 \leq a_r < 0 \\ 2a_r - a_r^2 & \text{if } 0 \leq a_r < 1 \\ 1 & \text{otherwise} \end{cases}, \quad \frac{\partial F(a_r)}{\partial a_r} = \begin{cases} 2 + 2a_r & \text{if } -1 \leq a_r < 0 \\ 2 - 2a_r & \text{if } 0 \leq a_r < 1 \\ 0 & \text{otherwise} \end{cases} \quad (2-27)$$

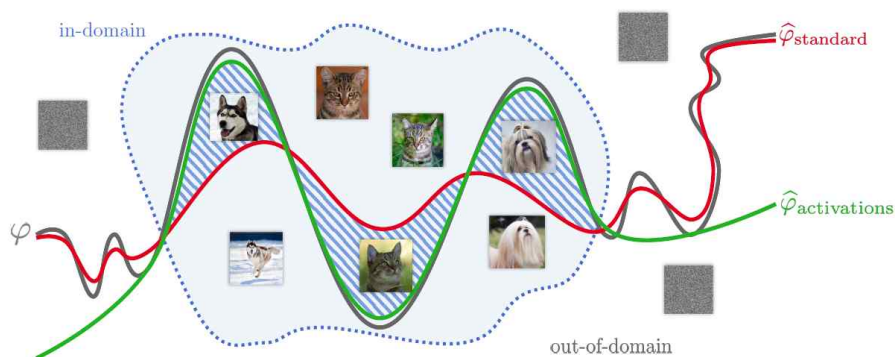
- 학습 파라미터들의 초기화 값 설정은 학습 시간과 성능에 크게 영향을 미침. 본 논문은 활성 함수를 변형하여 추정 Sign 함수를 사용하였는데 이와 가장 유사한 형태의 Clipped RELU를 기 학습 시킨 파라미터를 통해 학습 하여서 성

능 이득을 얻음.

제 3 절 지식 증류 기법(Knowledge Distillation)

1. 기술의 개요

- 지식 증류(Knowledge Distillation)란 다 계층의 모델로써 다수의 학습 파라미터로 대량의 연산을 통해 우수한 성능을 보인 teacher 모델에서 이보다는 경량화된 student 모델을 통해 비슷한 성능을 이끌어 낼 수 있도록 제안한 방법임.
- 모델 학습을 통해 결정된 클래스정보 보다는 완전 연결 계층(fully connected layer)의 출력으로 결정되는 각 클래스에 포함되어 있을 확률 분포를 활용하여서 보다 많은 정보가 선생님 모델에서 학생 모델로 전달(distillation)하는 방법이 사용됨 [Hinton2015]
- 이러한 방법을 고정 레이블(hard label)방법이 아닌 소프트 레이블(soft label)로 소개되기도 함



| cow | dog | cat | car | original hard targets |
|-----------|-----|-----|-----------|------------------------------|
| 0 | 1 | 0 | 0 | |
| cow | dog | cat | car | output of geometric ensemble |
| 10^{-6} | .9 | .1 | 10^{-9} | |
| cow | dog | cat | car | softened output of ensemble |
| .05 | .3 | .2 | .005 | |

[그림 2-22] 유사한 부류의 분류 (in-domain classification) 성능 향상에 긍정적인 영향을 미치는 soft label방법(Stock2020)

- [그림 2-22]에 소개된 소프트 레이블기법은 식 2-28에 나타난 것과 같이 Softer Softmax를 사용함. Softer Softmax는 기존의 Softmax 함수에 온도 (Temperature)라는 파라미터 T 를 추가하여서, T 가 높을수록 기존보다 더 soft한 확률분포를 얻을 수 있도록 만듦. 이는 증류 시 온도에 따라 증류물이 달라지는 것에 기인하여 이름 지어짐. Hinton (Hinton, 2015)의 논문에 따르면 이 T 값이 2~4일 때 증류(distillation)가 효과적으로 적용되었다고 기술되어 있음. $T=1$ 일 때 기존의 softmax함수와 동일한 함수가 됨.

$$q_i = \frac{\exp(\frac{z_j}{T})}{\sum_j \exp(\frac{z_j}{T})} \quad (\text{식 2-28})$$

- Hinton의 논문에서는 도메인 내의 분류 성능 향상을 위한 증류 방법으로 소프트 레이블과 고정 레이블을 한 방법을 더하여 합한 cross entropy모형이 더 정확한 성능을 보이는데, 고정 레이블을 취한 부분에 가중치를 더 적게 할수록 유사 항목 내의 분류 성능이 향상된다는 보고가 있음.
- 수식 (2-29)는 선생님 모델의 전체 클래스들의 확률 분포를 기반으로 한 p^g 와 학생 모델간의 KL divergence와 전체 클래스 중 특별히 혼돈이 쉬운 클래스 S^m 에 포함되는 집합 A^k 에 포함된 확률분포 p^m 대비 학생 모델 데이터 분포 q 간의 KL divergence를 고려한 조화를 이룬(ensemble) 추론 과정임. 이러한 두 종류의 확률 분포간 거리를 줄여가는 ensemble KL divergence 추론 과정을 통해 학생 모델은 혼돈이 쉬운 클래스 집합(유사 도메인 내의 분류)들 내의 분류 성능이 향상됨.

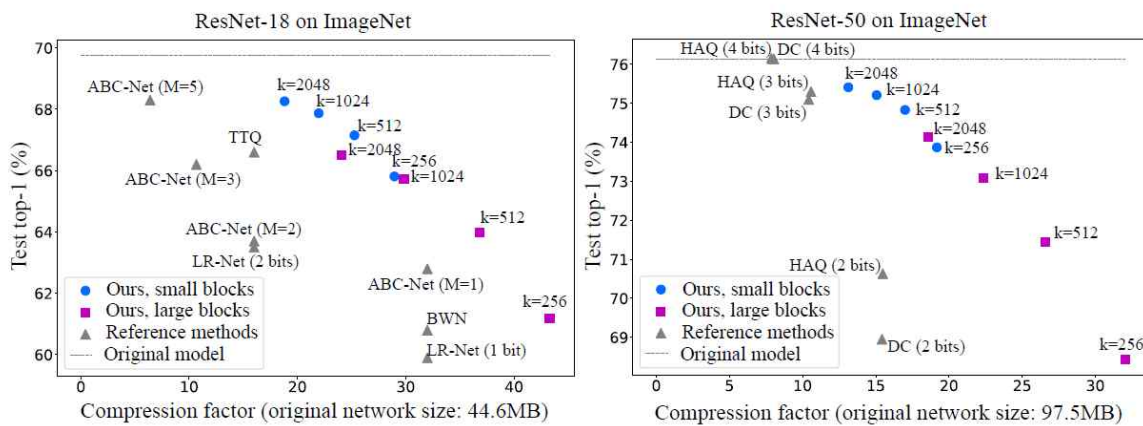
$$KL(p^g, q) + \sum_{m \in A_k} KL(p^m, q) \quad (\text{식 2-29})$$

2. Kill-the-bits [Stock2020]

- Kill-the-bits 논문에서는 순차적으로 지식 증류 기반 낮은 계층에서 높은 계층으로 계층별 양자화를 수행함. 양자화 기술에 고차원 벡터 공간을 데카르트 생산 부분 공간(Cartesian product subspace)로 분류하고 각 부분 공간을 양자화

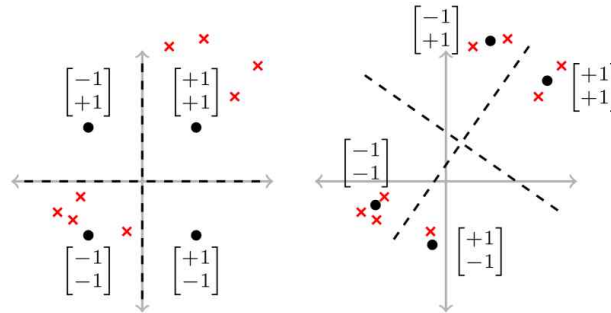
하는 PQ(Product Quantization) 기술이 사용됨. 각 계층별 부분 공간을 구성하는 코드 워드들을 특정 코드 북에 매핑하고 최적화 하는 과정이 기학습 모델 파라미터들을 선생님 모델로 하여 양자화 된 모델을 학생 모델로 보고 지식 증류과정을 통해 최적화가 수행됨. 모든 계층별 양자화와 코드 북 최적화 과정을 마치고 나면, 전체 모델에 대해 기 학습 파라미터를 선생님 모델로 두고 양자화 된 모델을 학생 모델로 둔 후 각 코드 워드의 코드 북에 대한 최적화 과정이 재 수행되는 과정을 통해 최적화가 진행됨.

- ImageNet 데이터 셋을 기준으로 압축률과 분류 정확도 성능 측면에서 [그림 2-23]를 통해 확인할 수 있듯이 (State-Of-ArT) SOTA 모델임. 그런데 학습시간 및 추론시간 측면에서는 선생님 모델에 대해 기 학습이 필요한 것에 더해 지식 증류로 양자화 하는 추가 시간이 ResNet50모델 기준 작은 블록을 이용하는 경우 대략 44시간이 소요됨. 이는 지식 증류 과정을 계층별로 수행하여 양자화를 최적화함과 동시에 전체 학습모델에 대해 지식 증류 과정을 통해 양자화를 최적화하는 과정에 기인함.

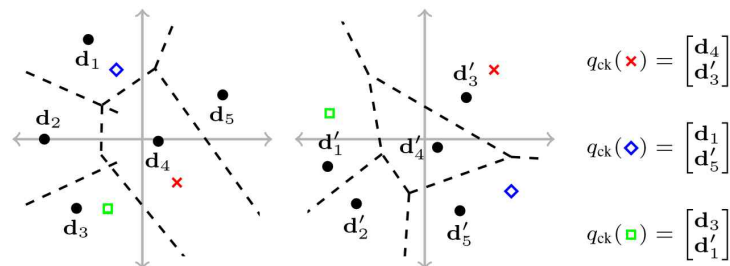


[그림 2-23] ImageNet 데이터기준 ResNet-18, ResNet-50모델에 대해 kill-the bits 모델의 압축률과 분류 정확도 성능 비교

- Product Quantization (PQ) 과정의 이해를 돕기 위해 유사한 형태의 알고리즘으로 [그림 2-23]와 [그림 2-24]에 표현된 Cartesian K-means 알고리즘과 Iterative Quantization 알고리즘을 들 수 있음.



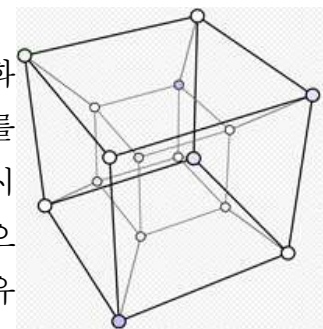
[그림 2-24] 2D data에 대한 0-k means기반 2종류의 quantization 예시 (Norouzi, 2013)



[그림 2-25] 4D data에 대한 Cartesian quantization예시

- [그림 2-24]와 [그림 2-25]에서 빨간 \times 는 데이터를, 검은색 점 \cdot 은 집합의 중심을 나타냄. [그림 2-24]의 좌편 그림은 2차원 데이터에 대해 각 집합의 경계가 좌표와 일치하는 점선으로 표현되어 있고 우편 그림은 회전, 늘임, 그리고 이동 과정을 통해 데이터의 위치들과 집합의 중심 간의 거리를 줄여 클러스터링(집합화) 과정의 효율성을 높임. [그림 2-25]는 4차원 데이터에 대해 Cartesian K-means를 반복과정의 처음과 마지막 과정을 왼쪽, 오른쪽 그림에 각각 표현한 것임. k-means 양자화 q_{ck} 는 반복 수행과정의 부분 공간들을 결합하여 4D 재건을 수행함.

- Iterative-Quantization은 코드 워드간의 거리를 최대화하고 연산의 단순성을 취하기 위해 모든 코드 워드를 “회전 하이퍼 큐브(hypercube - 오른쪽 그림의 예시와 같이 정육면체)의 꼭지점”에서 취하는 것을 원칙으로 함. 따라서 회전 하이퍼큐브로 결정지어진 제곱 유클리디안 거리로 결정지어진 코드 워드간의 거리는 해밍 거리(Hamming distance)¹⁾와 동일해짐 (Gong, 2011). D차원 하이퍼 큐브에서 코드 복은 수식 (2-30)와 같이 직교행렬(Orthogonal matrix)로 구성됨



$$\min_{R,a} \sum_x \|x - c(i(x))\|^2 \quad (2-30)$$

$$s.t. \mathbf{c} \in \{\mathbf{c} | R\mathbf{c} \in \{-a, a\}^D\}, R^T R = I$$

R은 직교행렬이고 I는 단위행렬이며, 코드북 C는 2^D 코드 워드를 포함함.

■ 일반적인 Product 양자화 과정은 다음과 같이 요약됨

- 가중치 벡터 $\mathbf{W} \in R^{C_{in} \times C_{out}}$ 를 m 개의 인접 부분벡터로 표현하고 코드 워드를 학습한 결과물은 $m \times C_{out}$ 행렬이 됨.
- m 개의 인접 부분벡터는 d ($d = C_{in}/m$)차원이 됨. 코드 북 $C = \{c_1, \dots, c_k\}$ 은 차원 d 의 k 개의 코드 워드로 구성됨.
- 가중치 벡터 \mathbf{W} 의 j -th 열 w_j 의 양자화버전 $q(w_j) = (c_{j1}, \dots, c_{jm})$
- 코드 북 학습 목적함수: $\|\mathbf{W} - \hat{\mathbf{W}}\|_2^2 = \sum_j \|w_j - q(w_j)\|_2^2$
- Product 양자화 과정을 통해 k^m 크기의 코드 북을 형성함

■ 이와 같은 일반적인 Product 양자화과정을 완전 연결 계층에서는 양자화 대상을 가중치가 아닌 마지막 출력 활성화함수에 초점을 맞추어 에러 함수를 수식 (2-31)과 같이 구성함. EM(Expectation Minimization과정) 반복을 통해 클러스터에 코드워드 할당 과정과 코드 워드(집합 별 중심)(M-step)를 갱신함.

$$\|y - \hat{y}\|_2^2 = \sum_j \|x(w_j - q(w_j))\|_2^2 \quad (2-31)$$

[EM]

① E step ~ Cluster의 member 할당

가중치의 각 column w_j 가 차원 d 를 갖는 m 개의 부분행렬로 나누어짐
각 부분행렬 v 는 code word c_j 에 할당됨

$$c_j =$$

② M step ~ code word update

$v_p (v_p \in I_c)$ 는 code word c 에 할당되는 부분 행렬을 나타냄

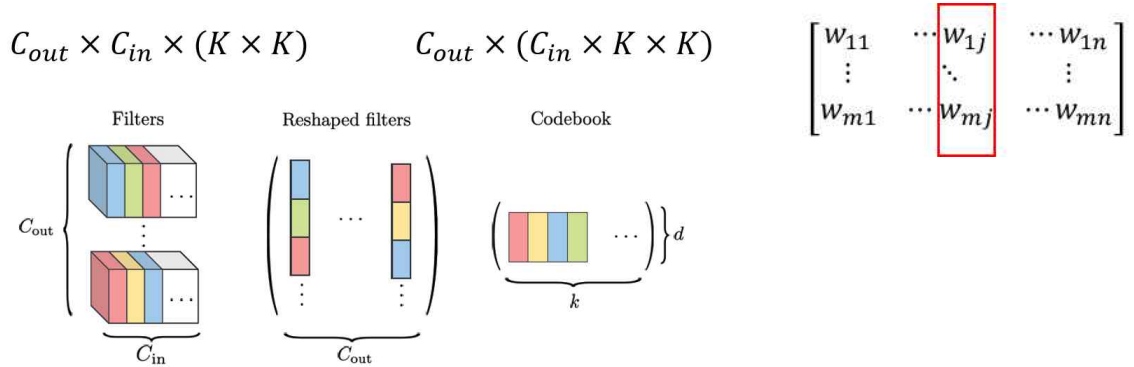
$$c^* =$$

- 코드 북은 양자화 하고자 하는 k 개의 벡터를 샘플링하고 이를 균일분포로 샘플링하여 초기화함. 집합 할당 시 부분 요소가 없는 집합을 고려하여

1) Hamming distance는 곱 집합위에 정의되는 거리 함수로, 대략 같은 길이의 두 문자열에서, 같은 위치에 서로 다른 기호들이 몇 개인지 세어 계산됨

집합중 가장 많은 요소가 있는 집합주변에 집합을 추가 할당하여서 부분 요소가 없는 집합(empty cluster)문제를 해결함.

- 컨벌루션 계층에서는 4D 가중치 벡터에 대해 벡터 간 상관관계가 높을수록 양자화를 통한 이득을 높이는 것을 목적으로 벡터 간 상관관계를 크게 구성함. 따라서 한 계층의 가중치 벡터 $W \in R^{C_{out} \times C_{in} \times K \times K}$ 에 대해 입력 채널과 커널을 묶어 계산한 각 행렬의 열별 행 C_{out} 을 재구성하여 코드북에 매핑함



[그림 2-25] 컨벌루션 계층의 코드 북 매핑을 위한 필터 재구성

```

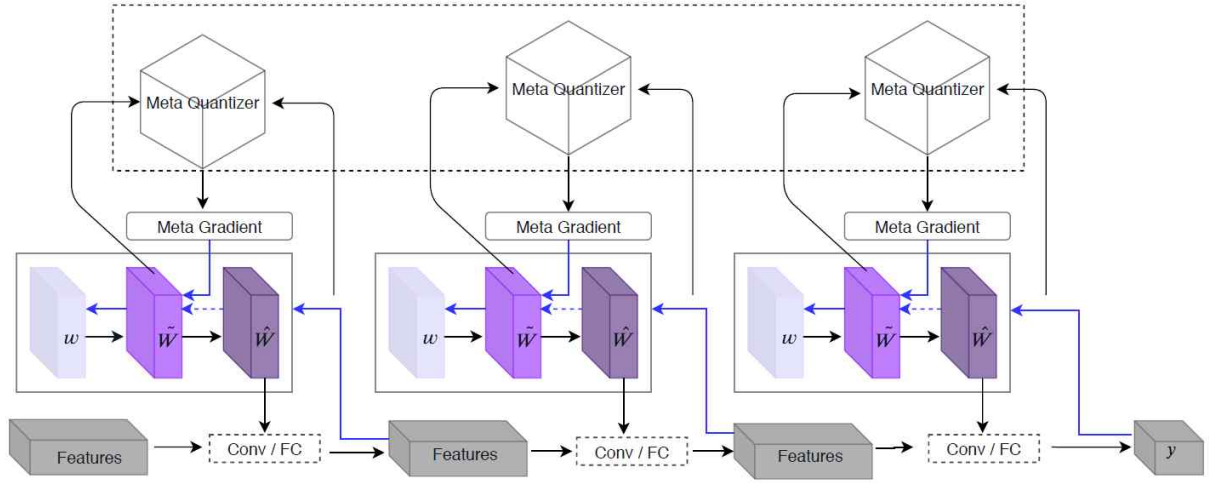
Def reshape_weight(weight):
    if len(weight.size()) == 4:
        C_out, C_in, k, k = weight.size()
        return weight.view(C_out, C_in * k * k).t()
    else:
        return weight.t()

```

제 5 절 하이브리드 및 기타 기법

1. MetaQuant (Chen2019b)

- STE가 가지고 있는 한계점으로 양자화 된 가중치를 통해 미분 연산이 불가능하여 실제 가중치가 아닌 추정된 가중치를 통해 연산을 하여 발생하는 손실을 줄이고자 하는 방법으로 미분을 연산하는 Meta Quantizer를 따로 두어 연산하는 방법임.



[그림 2-26] Meta Quantizer 동작 개괄

- Meta Quantizer를 적용한 모델로 양자화 초기 모델인 Binary-Weight(BWN) 모델과 DorefaNet이 있음. Meta Quantizer를 적용한 BWN 모델과 DorefaNet의 모델은 수식 (2-32)와 (2-33) 같음

- DorefaNet과 BWN의 전처리함수를 A라하고 전 처리된 가중치를 \tilde{W} 라 가정, 양자화 함수를 Q, 그리고 양자화된 가중치를 \hat{W} 라 하면, forward연산을 적용한 MetaQuantizer함수는 수식 (2-32)와 (2-33) 각각 임.

$$\text{DorefaNet 전처리 : } \tilde{W} = A(W) = \frac{\tanh(W)}{2\max(|\tanh(W)|)} + \frac{1}{2} \quad (\text{식 2-32})$$

$$\text{DorefaNet 양자화 : } \hat{W} = Q(\tilde{W}) = 2 \frac{\partial [(2^k - 1) \tilde{W}]}{2^k - 1} - 1 \quad (\text{식 2-33})$$

$$\text{BWN 전처리 : } \tilde{W} = A(W) = W \quad (\text{식 2-34})$$

$$\text{BWN 양자화 : } \hat{W} = Q(\tilde{W}) = \frac{1}{n} \|\tilde{W}\|_{l_1} \times \text{sign}(\tilde{W}) \quad (\text{식 2-35})$$

- 미분 연산 과정은 다음 과정임

$$\text{양자화된 가중치의 미분 값 : } g_{\hat{W}} = \frac{\partial l}{\partial \hat{W}} \quad (\text{식 2-36})$$

전처리 가중치와 양자화된 가중치를 고려한 meta-미분 값 :

$$g_{\tilde{W}} = M_{\Phi}(g_{\hat{W}}, \tilde{W}) \quad (\text{식 2-37})$$

32bit 가중치에 대한 미분 값:

$$g_W = \frac{\partial l}{\partial \tilde{W}} \frac{\partial \tilde{W}}{\partial W} = g_{\tilde{W}} \frac{\partial \tilde{W}}{\partial W} = M_{\Phi}(g_{\hat{W}}, \tilde{W}) \frac{\partial \tilde{W}}{\partial W} \quad (\text{식 2-38})$$

$$\text{DorefaNet 양자화 미분 : } \frac{\partial \tilde{W}}{\partial W} = \frac{1 - \tanh^2(W)}{\max(|\tanh(W)|)} \quad (\text{식 2-39})$$

$$\text{BWN 양자화 미분 : } \frac{\partial \tilde{W}}{\partial W} = 1 \quad (\text{식 2-40})$$

최적화 함수와 함께 계산되어지는 미분 정제:

Stochastic Gradient Descent (SGD) 기반의 미분 정제 함수:

$$\pi(g_W) = g_W \quad (\text{식 2-41})$$

Adam 기반의 미분 정제함수:

$$\pi(g_W) = g_W \quad (\text{식 2-42})$$

32bit 가중치 업데이트 연산 법:

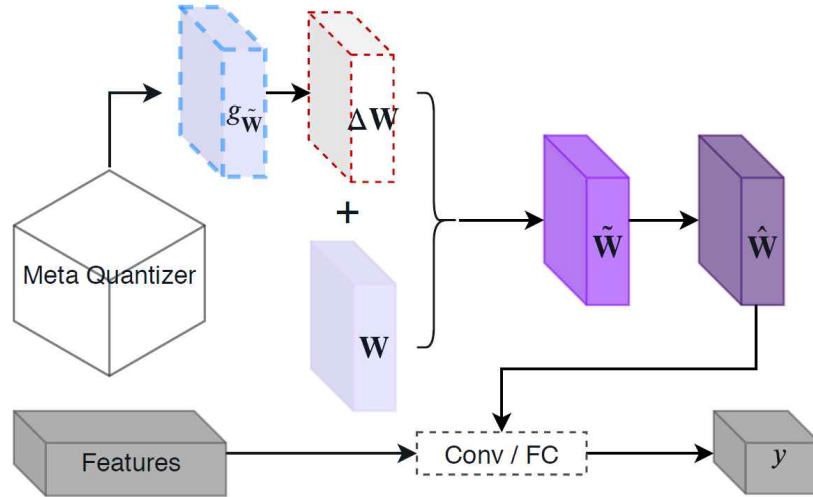
$$W^{t+1} = W^t - \alpha \pi(g_W^t) \quad (\text{식 2-43})$$

- Meta Quantizer의 Meta 미분 연산과정은 각 가중치의 미분 연산을 보조하도록 독립적으로 추가 계산되는 [그림 2-27]과 같은 뉴럴 네트워크 구조를 지님. 그림으로 빨간 상자로 표현된 연산과정을 통해 양자화 된 미분 값에 optimizer 계산을 통한 미분 값과 learning rate α 연산이 포함됨.

MetaQuantizer를 기반으로 한 순 전파(Forward propagation), 역전파(Backward propagation) 연산은 각각 식 2-44, 2-45임

MetaQuantizer 순전파 연산:

$$\tilde{W}^t = A(W^t) = A \left[W^{t-1} - \alpha \times \pi \left(M_{\Phi}(g_{\tilde{W}}^{t-1}, \tilde{W}) \frac{\partial \tilde{W}^{t-1}}{\partial W^{t-1}} \right) \right] \quad (\text{식 2-44})$$



[그림 2-27] meta quantizer를 quantization 학습 과정 중에 도입

MetaQuantizer 역전파 연산:

$$\frac{\partial L}{\partial \Phi^t} = \frac{\partial L}{\partial \tilde{W}^t} \frac{\partial \tilde{W}^t}{\partial \Phi^t} = M_{\Phi}(g_{\tilde{W}}^t, \tilde{W}^t) \frac{\partial \tilde{W}^t}{\partial \Phi^t} \quad (\text{식 2-45})$$

- Meta Quantizer를 적용하여 성능 향상을 CIFAR10 모델을 통해 확인 가능하나 양자화 목적에 맞지 않는 모델 구성임. 양자화를 적용하는 목적은 모델의 경량화 하는 것에 그 뜻이 있으나, Meta Quantizer를 통해 추가 모델을 동시에 같이 연산하는 형태가 되어 그 연산 량이 학습 시간에 3배에 이르도록 학습 파라미터가 증가하는 단점이 있는 모델임.

제3장

경량 네트워크 구조 기법

제 1 절 CNN 모델의 개요

1. CNN 모델의 배경

□ CNN 역사적 배경

- CNN은 1980년대 등장한 완전 연결 뉴럴 네트워크(fully connected neural network)의 성능의 한계를 극복하기 위해 1998년 Yann Lecun이 제안한 모델 ReNet-5가 시초임 [Yan1998a]. 당시 성능은 mnist 데이터 셋(32x32 이미지, 10 클래스)에 대해 60,000 학습 데이터에 대해 99.2% 정확도를 확인함.
- 2012년 ILSVRC 경시대회에서 Alex Krizhevsky의 CNN을 기반으로 한 AlexNet이 우승을 차지하며 뉴럴 네트워크는 다시 주목을 받기 시작함 [Alex 2012a] ReNet-5 모델과 AlexNet은 유사한 네트워크 구조를 떠나 더 깊은 레이어로 구성되고 활성화 함수로 SIGMOID 함수를 대체한 ReLU 함수를 이용했다는 것과 그래픽 카드를 이용한 대용량 행렬 연산이 활용가능해지면서 256x256 이미지, 1000개의 클래스에 대해 기존의 74% 정확률과 큰 격차를 보이며 84%의 정확도를 보이며 큰 주목을 받게 됨.
- 이후 Vision 계통에서 딥러닝 알고리즘이 큰 관심을 받게 되었고, 2015년 ResNet은 인간의 이미지 인식 오류를 5.1%를 넘어섬 [He2016a].

□ Break-Through 모델 AlexNet의 구성

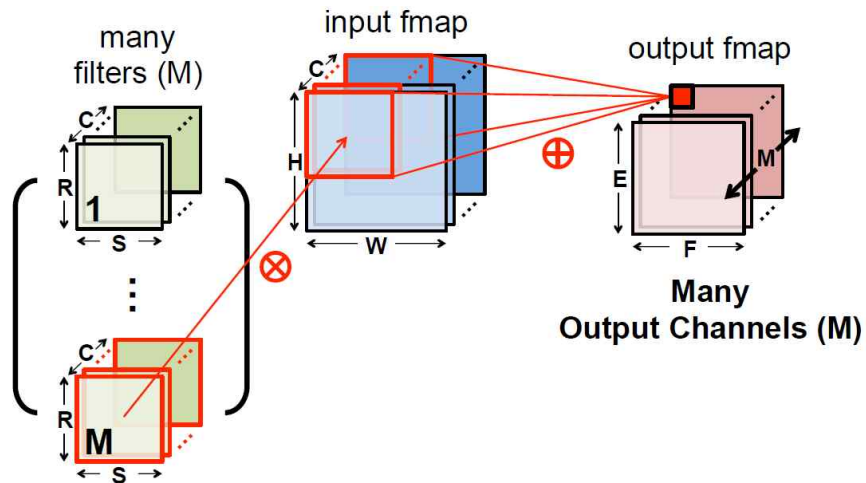
- AlexNet의 모델은 5개의 컨벌루션 계층과 2개의 완전 연결 계층으로 구성됨. 256x256 입력 이미지에 대해 깊은 층 연산(deep learning) 과정을 통해 각 이미지들의 특징을 드러내는 특징들이 추출 되는 과정(Convolution)과 이를 요약(pooling)하는 과정을 거쳐 각 구성 이미지를 1000개의 클래스 중 클래스를 분류(softmax)하는 계층으로 구성됨. AlexNet의 구성은 표 3-1과 같음

<표 3-1> AlexNet 각 계층의 구성

| Layer 이름 | Kernel 크기 (pooling 크기) / stride | Output Size (# of Out Channels + Feature Map) | # of parameters | # of FP operations |
|----------|---------------------------------|---|-----------------|--------------------|
| conv1 | 11×11 / 4 | 96 × 55 × 55 | 35K | 55G |
| pool1 | 3 × 3 / 2 | 96 × 27 × 27 | | |
| conv2 | 5 × 5 / 1 | 256 × 27 × 27 | 614K | 227G |
| pool2 | 3 × 3 / 2 | 256 × 13 × 13 | | |
| conv3 | 3 × 3 / 1 | 384 × 13 × 13 | 885K | 65G |
| conv4 | 3 × 3 / 1 | 384 × 13 × 13 | 1.3M | 98G |
| conv5 | 3 × 3 / 1 | 256 × 13 × 13 | 885K | 65G |
| pool3 | 3 × 3 / 2 | 256 × 6 × 6 | | |
| fc1 | | 4096 | 37M | 74M |
| fc2 | | 4096 | 16M | 32M |
| fc3 | | 1000 | 4M | 8M |
| softmax | | 1000 | | |

2. CNN 모델의 일반적인 연산 기법

- 컨벌루션 계층은 이미지의 2D 공간상의 특징(Feature)을 추출하는 필터로 구성됨. 이러한 연산과정은 이미지의 넓이와 높이 방향의 인접 픽셀들에 대한 필터를 학습하게 되고, 2D 공간의 같은 위치의 모든 채널에 대해 선형 결합 연산을 수행함. 즉, 하나의 커널은 입력 영상의 각 위치별 반응치를 계산하여 2D 반응 맵을 출력하고, 각 위치별 모든 채널 픽셀에 대한 선형 결합을 수행함.



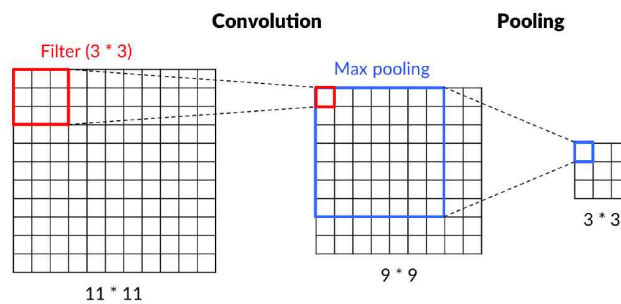
[그림 3-1] CNN 모델 구조

- 컨벌루션 연산은 [그림 3-1]에 정의된 것과 같이 입력 이미지의 너비와 높이를 H, W 라 하고, 한 장의 필터 크기가 $R \times S$ 인 C 개의 필터가 있을 때, 출력 특징 맵과 입력 특징 맵의 크기가 같다고 가정하고 출력 특징 맵의 개수를 M 이라 할 때, 컨벌루션 연산량은 다음과 같음

- Parameter 수 : $R \times S \times C^2 \times M$
- 연산량 : $R \times S \times C^2 \times H \times W \times M$
- Memory access cost : $R \times S \times C \times M + (H \times W) \times (C + M)$

3. CNN 모델 연산의 특징

- 컨벌루션 연산에 있어 큰 특징 맵을 컨벌루션 입력으로 가지는 것이 유리하지만 CNN에서는 연산 비용을 고려하여 대부분 부분 샘플링을 수행함



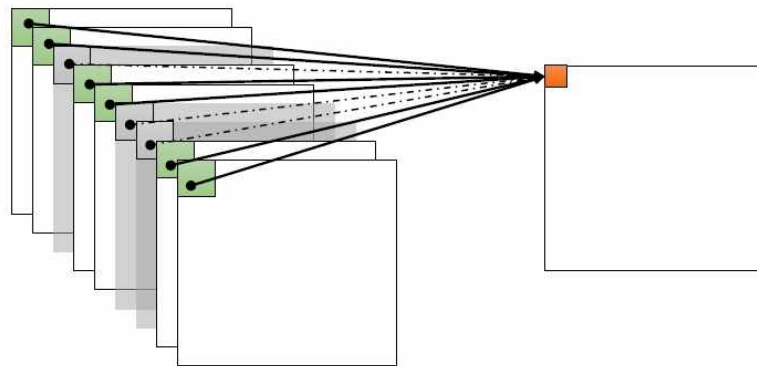
[그림 3-2] Convolution연산의 Filter와 pooling 의 예시

- 각 계층(layer)별로 채널의 수를 늘릴수록 더 많은 이미지 특징을 드러내는 학습이 가능하나, 채널수의 증가는 학습 파라미터의 수 증가, 이에 따른 학습 시간 증가 그리고 주어진 데이터에 over-fitting(과적합) 문제를 야기함.

| Group name | Output size | ResNet | Wide ResNet | PyramidNet |
|------------|-------------|----------------------------------|----------------------------------|----------------------------------|
| conv1 | 32x32 | 3x3, 16 | 3x3, 16 | 3x3, 16 |
| conv2 | 32x32 | 3x3, 32 3x3, 32 3x3, 32 | 3x3, 64 3x3, 64 3x3, 64 | 3x3, 48 3x3, 69 3x3, 96 |
| conv3 | 16x16 | 3x3, 64 3x3, 64 3x3, 64 | 3x3, 128 3x3, 128 3x3, 128 | 3x3, 122 3x3, 149 3x3, 176 |
| conv4 | 8x8 | 3x3, 128 3x3, 128 3x3, 128 | 3x3, 256 3x3, 256 3x3, 256 | 3x3, 202 3x3, 299 3x3, 256 |

[그림 3-3] ResNet, WideResNet 그리고 PyramidNet

- 신경망 학습 과정 중, 출력 결과에 영향을 거의 주지 않는 간선 즉 가중치 값이 0에 가까운 노드간의 연결이 존재함. 이는 보통 가지치기(pruning) 기법을 통해 신경망의 연산량과 파라미터 수를 줄이는데 사용됨. 유사한 문제점이 불필요한 채널 단위로 발생하기도 함.
- 좁게는 하나의 필터에서 넓게는 전체 신경망에 불필요한 채널이 발생할 수 있음. 이러한 불필요한 채널일지라도 가중치를 가지고 있고 연산 시간을 소모하게 되며 이러한 채널을 찾는 것을 뉴런 가지치기(Neuron Pruning)를 이용하더라도 제거 여부 판단에 어려움이 있음.



[그림 3-4] Pruning의 예시

- 각 필터는 입력 영상의 모든 채널에 사용되나, 모든 채널-필터 쌍이 높은 상관관계를 가질 수 없음. 결과적으로 이는 성긴 상관관계 행렬(sparse correlation matrix)을 보이게 되며, 이는 학습 수렴 속도 저하를 가져오는 불필요한 가중치와 연산이 존재한다고 간주할 수 있음.

제 2 절 CNN 모델의 경량 네트워크 구조

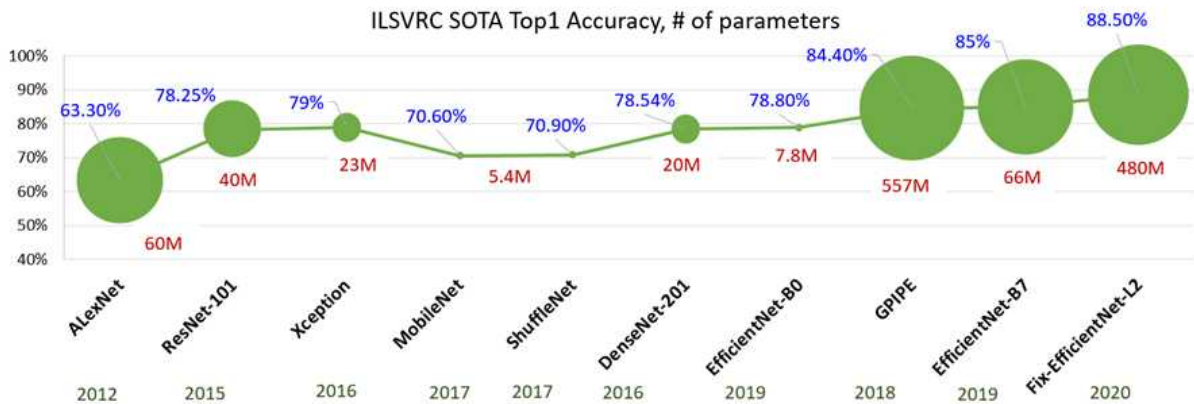
- 2012년 이후 2019년에 이르기까지 뉴럴 네트워크의 구조 변경으로 모델 파라미터 개수를 줄이면서도 성능을 향상시키기 위한 최적 모델 설계 모델들이 있음. Top1 정확도 기준으로 2012년 AlexNet 대비 2019년 “EfficientNet-B0”는 파라미터수가 8배 줄고, 정확도는 15.5% 향상됨.

<표 3-2> 파라미터수와 정확도를 고려한 뉴럴 네트워크 아키텍처의 특징

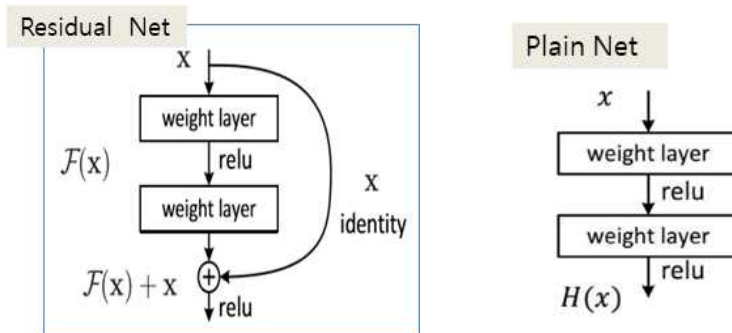
| 모델 | 특징 |
|---------------------|--|
| ResNet | Residual Connection, Bottleneck |
| Xception, MobileNet | Depthwise Separable Point-wise convolution |
| AlexNet, ShuffleNet | Grouped Convolution |
| ShiftNet | Shift Convolution |
| DenseNet | Dense Convolution |
| EfficientNet | Compound Scaling |

1. Residual Connection, Bottleneck[He, 2016a, 2016b]

- 잔류 연결(Residual Connection)은 일반적으로 계층들의 출력 특징 맵을 활성화 함수를 통과시킨 결과인 $H(x)$ 를 그대로 사용하는 것과 다르다. 즉, 출력을 입력 데이터와 활성화함수를 통과시킨 값의 차인 잔류 $F(x) = H(x) - x$ 값으로 받되, 입력 값을 연산 후에 더하는 동일연결(identity connection)을 일컬음

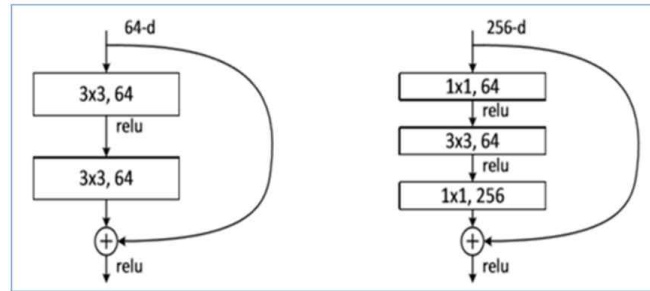


[그림 3-5] parameter수와 accuracy를 고려한 Neural Network Architecture의 변화



[그림 3-6] Plain Network과 Residual block

- 병목 연결(Bottleneck Connection)은 차원수와 파라미터수 감소를 위해 연달은 3x3 컨벌루션을 대체하여 하나의 3x3 전후에 1x1 컨벌루션을 배치한 병목 형태의 연산



[그림 3-7] all-3x3 bottleneck(for ResNet50/101/152)

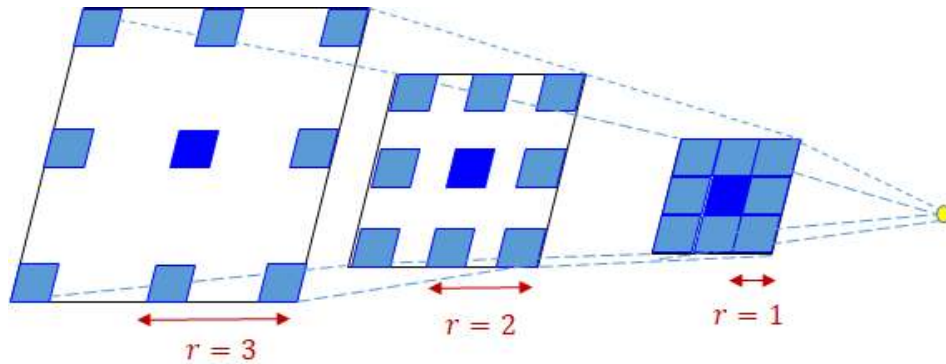
<표. 3-3> bottleneck 모델 적용에 따른 총 연산량 감소

| 모델 | 연산 | 총 연산량 |
|------------|---|-------|
| all-3x3 | $(28 \times 28 \times 64) \times (3 \times 3 \times 64) \times 2$ | 57M |
| bottleneck | $(28 \times 28 \times 256) \times 64$ $+ (28 \times 28 \times 64) \times (3 \times 3 \times 64)$ $+ (28 \times 28 \times 64) \times 256$ $= 14M + 25M + 12M$ | 51M |

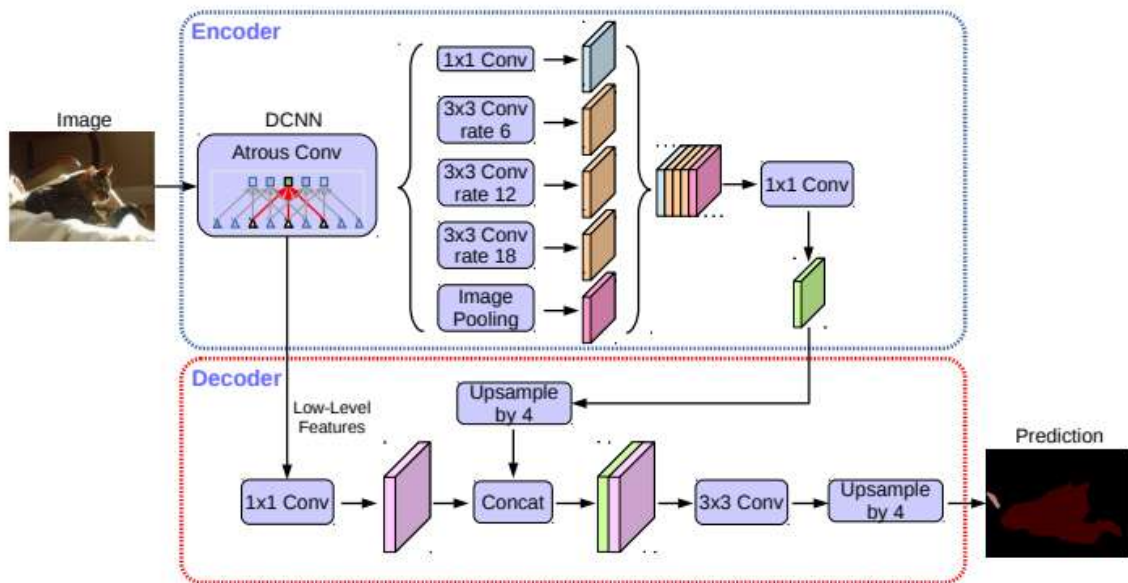
2. Dilated Convolution

- 팽창 컨벌루션(Dilated Convolution)은 수용 영역(Receptive Field²⁾)의 크기를 늘려가며 입력 영상의 특징들을 추출하는 방법으로, 수용 영역의 크기는 문맥의 정보를 얼마나 크게 잡는지에 따라 달라짐. 그림 3-8은 비율을 $r=1$, $r=2$, $r=3$ 으로 하여 팽창 컨벌루션 연산 공간을 (3x3, 5x5, 7x7)로 확장하면서 수용영역을 확장해 가는 것을 표현함. 일반 컨벌루션과의 차이는 비율에 따라 수용 영역의 크기는 커지지만 커널의 크기는 3x3으로 유지되는 점임

2) receptive filed는 출력 뉴런 하나에 영향을 미치는 입력 뉴런 공간의 크기를 일컫는다.



[그림 3-8] 팽창 컨벌루션 (Atrous Convolution) (Fu, 2017)



[그림 3-9] DeepLab3에서 이용한 팽창 컨벌루션 (Chen, 2018)

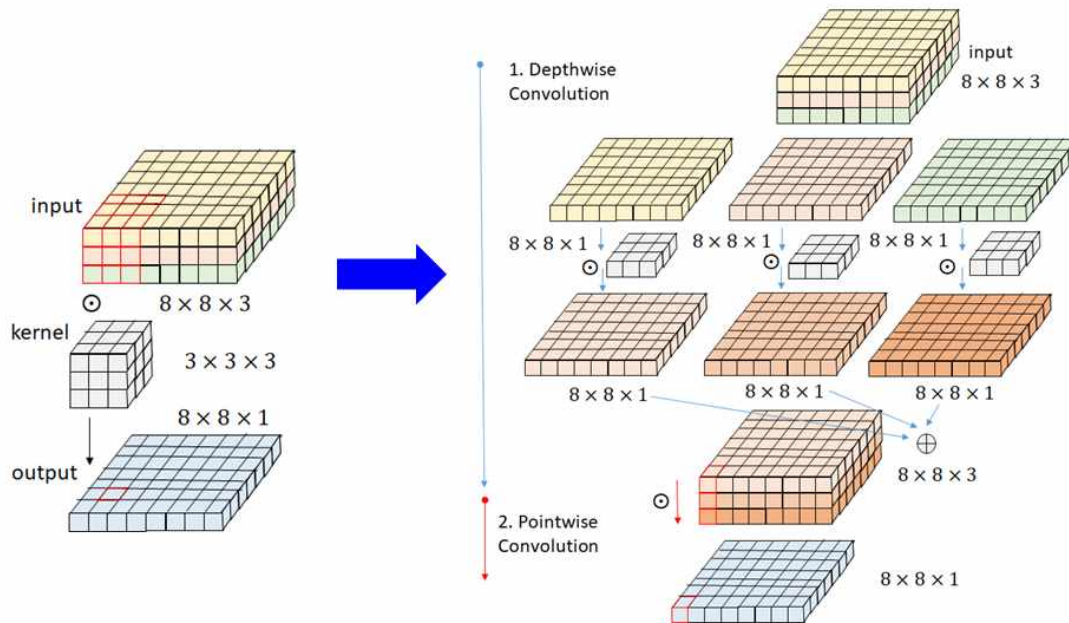
- [그림 3-9]는 통합(pooling)이후 분류(unpooling)을 할 경우 (인코딩 이후 디코딩을 할 경우) 성겨진 이미지의 영상 보간을 위해 팽창 컨벌루션의 비율을 점차적으로 늘려 피라미드 형태로 쌓은 Atrous Separable Spatial Pyramid Pooling(ASSPP)을 차용한 DeepLabV3의 구조 그림임.

3. Depthwise Separable Convolution MobileNet [Howard2017a]

- 일반적으로 컨벌루션 연산은 공간(spatial) 연산과 채널(channel)별 연산을 한 번에 진행함. 이를 공간 연산 후에 채널별 연산을 순차적으로 진행하여 연산량

을 줄이는 방법.

- 채널의 수 C , 입력 영상 크기 (H, W) , 커널 필터의 크기 (R, S) , 필터 출력 채널수를 M 이라 할 때, 표 3-4에서 보이는 바와 같이 Depth-wise separable 컨볼루션 연산은 연산 및 파라미터의 감소 효과를 제공할 수 있음



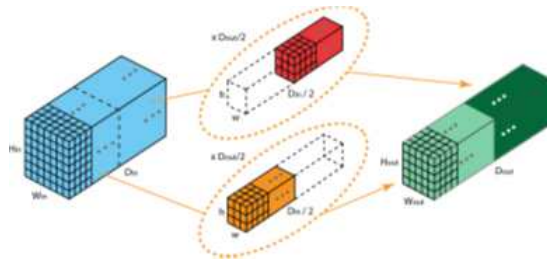
[그림 3-10] DepthWise Separable Point-Wise Convolutions

<표. 3-4> Depthwise Separable PointWise 컨볼루션에 따른 연산, 파라미터수 감소 계산

| | Spatial 컨볼루션 | D-wise Separable 컨볼루션 |
|-----------------------|---|---|
| 연산 수 | $M \times (R \times S \times C + 1)$ $= M \times R \times S \times C + M$ | $C \times (R \times S + 1) + M \times (C + 1)$ $= C \times R \times S + M \times C + C + M$ |
| parameter수 | $R \times S \times C \times H \times W \times M$ | $C \times H \times W \times R \times S + C \times H \times W \times M$ $= C \times H \times W \times (R \times S + M)$ |
| FLOPS (256 kernel) | $(3 \times 3 \times 3 \times 256) \times (8 \times 8)$ $= 442,368 \text{ FLOPs}$ | 1. Depthwise 컨볼루션 $(3 \times 3 \times 3) \times (8 \times 8) = 1,728$ 2. Pointwise 컨볼루션 $(1 \times 1 \times 3 \times 256) \times (8 \times 8) = 49,152$ $1,728 + 49,152 = 50,880 \text{ FLOPs}$ |

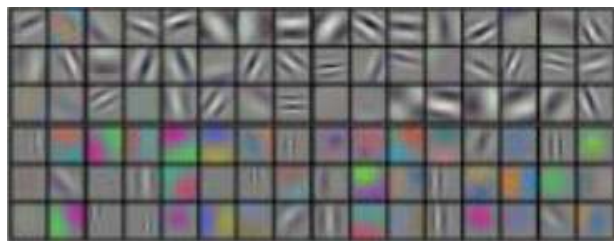
4. Grouped Convolution (그룹별 컨벌루션)

- 그룹별 컨벌루션은 필터 내 채널 정보를 그룹지어 분리하여 학습하여 병렬처리에 유리하도록 디자인. 필터 그룹별 상관관계가 높은 정보를 학습하여서 성기도록 (Sparse 하도록) 디자인하여 더 적은 파라미터들을 갖는 유리한 디자인임.



[그림 3-11] Group Convolution

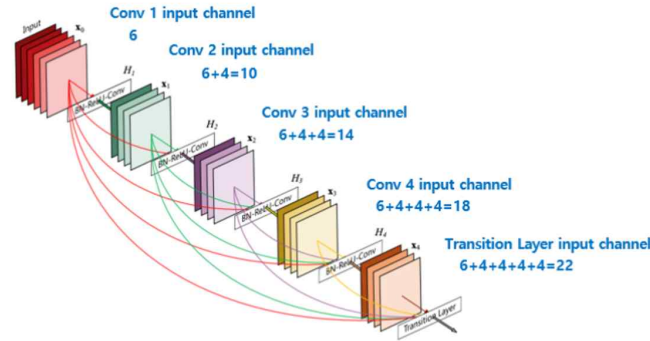
- 그룹 별로 연산된 커널들은 서로 다른 특징을 드러냄. AlexNet(Alex, 2012a)도 그룹별 컨벌루션을 수행하였는데, 아래 그림과 같이 그룹별로 서로 다른 커널 학습이 구별 지어지는 것을 확인.



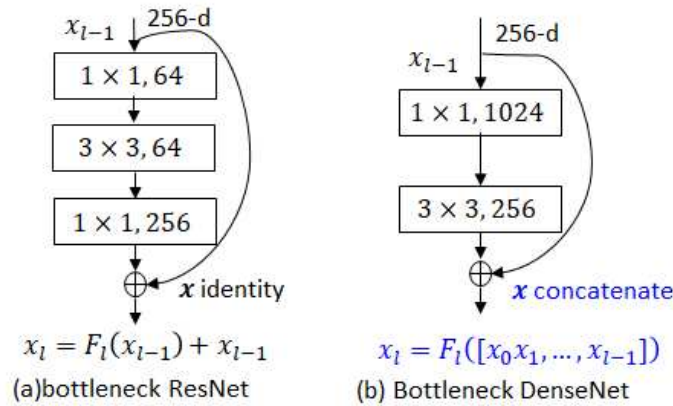
[그림 3-12] (위) : 음영위주의 커널 학습 그룹,
(아래) : 색상과 패턴위주의 학습 그룹

5. Dense Convolution [Huang2017a]

- 기존의 특징 맵을 더하는(additive) 형태가 아닌 쌓아가는(concat) 형태로 취합 (기존 특징 맵을 재사용 연산). 변화 계층(transition layer)에서 특징 맵을 압축하여 적은 파라미터수로 더 나은 성능을 확인



[그림 3-13] DenseNet



[그림 3-14] ResNet과 DenseNet (additive구성 vs. concat 구성)

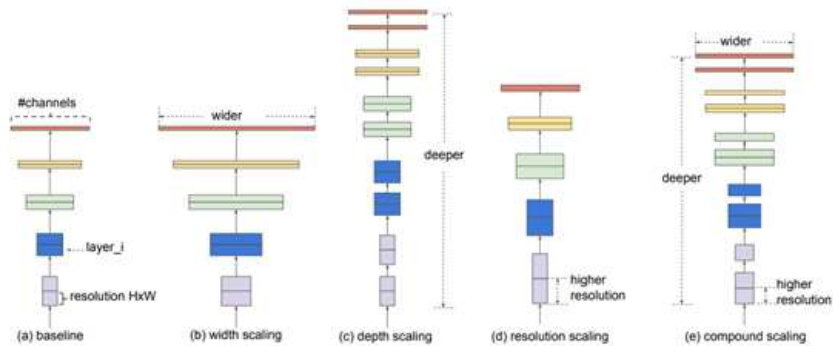
6. Compound Scaling

- Auto ML 기법으로 강화학습 기법으로 최적의 딥러닝 모델을 찾는 모델로 NAS, NASNet, mNASNet을 대표할 수 있음. EfficientNet은 mNASNet을 통해 찾은 모델을 기본 모델로 함.
- NAS(Neural Network Architecture Search)는 800개의 GPU (nVidia K40)로 28일에 걸쳐 CIFAR10을 최적화 시키는 모델을 찾아냄 [Zoph2017a].
- NASNet은 500개의 GPU(nVidia P100)을 이용하여 4일에 걸쳐 최적의 모델을 찾음. 학습 시간 단축을 위해 탐색 범위를 좁혀서(블록 단위로 두 개의 은닉 계층, 연산 방법 그리고 은닉 상태를 결합하는 방법을 결정) 네트워크 구조를 탐색하게 한 것이 주요 특징임 [Pham2018a].

- mNASNet은 최적의 모델을 찾기 위해 확인하는 것으로 모델의 정확도 외에 on-Device의 지연(latency)을 고려요소로 넣음 [Tan2018a] [EfficientNet] [Chen2019a]
- mNASNet으로 찾은 모델을 기본 모델 B0로 삼고, CNN 모델의 3요소 (깊이 - 계층 α^ϕ , 너비-채널 β^ϕ , 해상도-입력 이미지크기 γ^ϕ)의 효율적인 확장 방법을 찾은 것으로, 다음 수식을 만족하는 $\alpha = 1.2$, $\beta = 1.1$, $\gamma = 1.15$ 를 찾아냄. 이 값의 $\alpha = 1.2$, $\beta = 1.1$, $\gamma = 1.15$ 를 고정한 채, ϕ 를 늘려 B1~B7 모델을 탐색함.

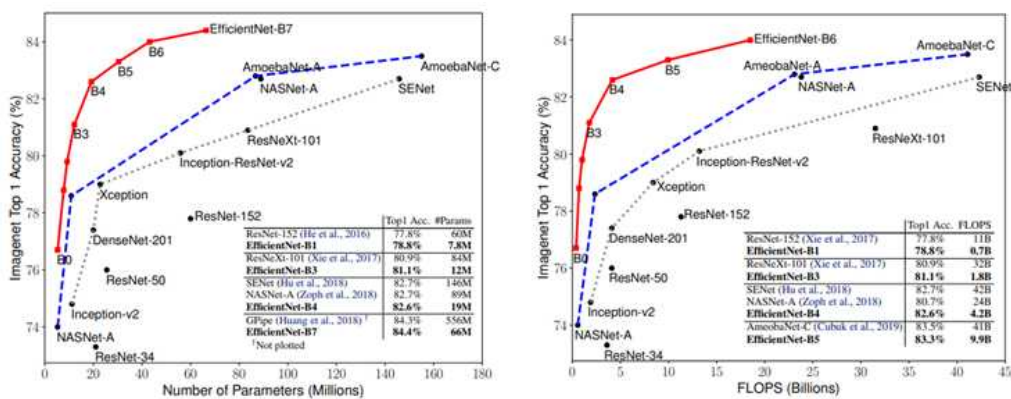
$$\alpha^1 \cdot \beta^2 \cdot \gamma^2 \approx 2 \quad (1)$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$



[그림 3-15] 깊이(layer), 너비(채널), 해상도(입력이미지크기) 변화에 따른 모델 탐색

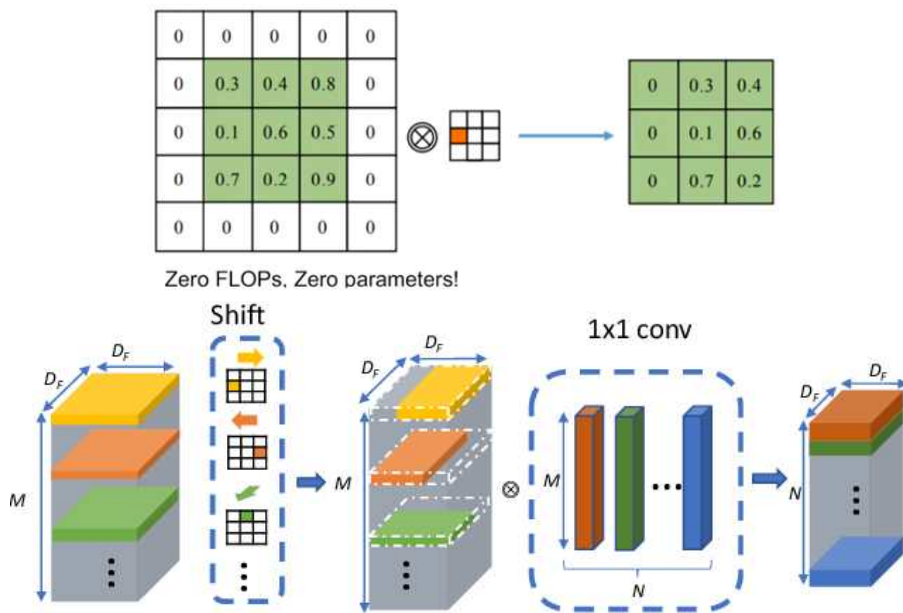
- EfficientNet은 타 모델 대비 파라미터수를 8~9배 줄이면서, 추론 속도 성능은 6배 정도 향상시킴.



[그림 3-16] ImageNet 데이터셋 (좌) 모델 크기 vs. 정확도 (우) FLOPS vs. 정확도

7. Shift Convolution

- 공간 컨벌루션을 Shift 연산으로 대체하여 컨벌루션 연산을 Shift 연산과 1x1 Pointwise 채널 컨벌루션으로 구성하여 연산을 줄임. ShiftNet 컨벌루션 연산을 통해 정확도 향상과 파라미터 수 감소뿐 만 아니라 학습시간을 줄임
- Shift 컨벌루션은 3x3 필터의 한 픽셀만 1인 값을 갖는 경우 컨벌루션 결과가 shift된 모습으로 연산 없이 메모리 내 위치 이동(memory shift)로 대체가능함.



[그림 3-17] Shift 연산 후 1x1 point-wise convolution으로 구성한 ShiftNet

- Shift 연산은 다음과 같이 정의됨:

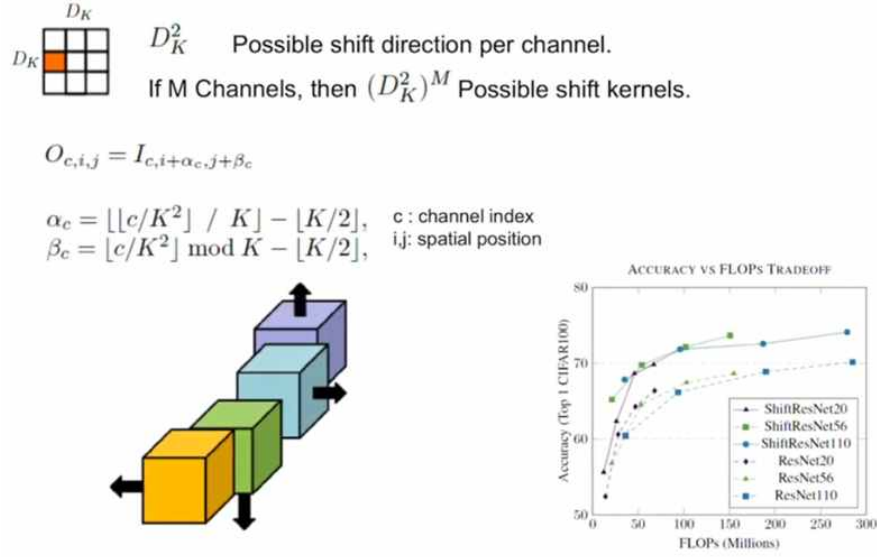
$$O_{c,i,j} = I_{c,i+\alpha_c,j+\beta_c}$$

I 는 입력 특징 맵을 나타내고, O는 출력 특징 맵을 나타냄.

c는 채널 인덱스를 나타내며 i, j는 각각 공간의 x좌표와 y좌표 위치를 표현함.

α_c, β_c 각각은 좌우, 상하 shift 크기를 표현함.

- ShiftNet중 그룹별 shift는 모든 채널이 같은 방향으로 Shift 되어 연산 성능을 떨어뜨리는 것을 방지하기 위해 채널 내에 그룹별 Shift 방향을 지정하여 연산을 시도함 [Wu2017]



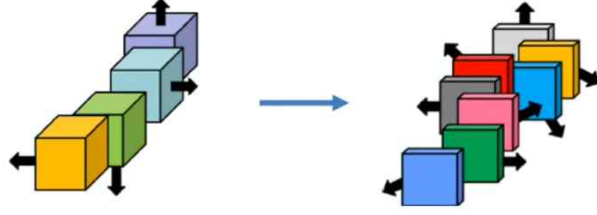
[그림 3-18] group shift operation

- ShiftNet 중 Active Shift는 모든 채널별로 최적의 Shift값을 찾도록 디자인한 모델로 Shift연산 결과에 대해 역전파에서 미분 연산이 가능함, 즉 미분 가능함 [Jeon, 2018].

$$O_{c,i,j} = \sum_{(n,m) \in \Omega} I_{c,n,m} \cdot (1 - |i + \alpha_c - n|)(1 - |j + \beta_c - m|)$$

$\Omega \ni (n,m)$ 는 $(i + \alpha_c, j + \beta_c)$ 의 네 지점 이웃 integer 위치 집합을 표현함.

- shift값(상하, 좌우 shift 각각 α, β)이 정수 값이 되어야하는 제약 사항을 Bilinear 보간법을 활용함. 즉 필터사이즈 변경이 bilinear 보간으로 보정되어 Shift 연산 수행.



$$S_G(X) = \begin{bmatrix} X_{1:n,:}^1 \\ X_{n+1:2n,:}^2 \\ \dots \\ X_{(G-1)n+1:, :}^G \end{bmatrix} \quad S_C(X) = \begin{bmatrix} X_{1:n,:}^1 \\ X_{n+1:2n,:}^2 \\ \dots \\ X_{C:, :}^C \end{bmatrix}$$

Formulate shift value(α_c, β_c) as a learnable additional parameters!

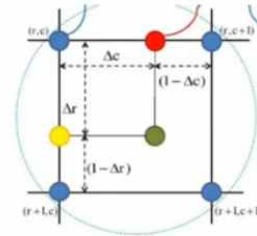
[그림 3-19] Active Shift operation

Calculate value through Bilinear Interpolation

$$\begin{aligned}\bar{x}_{c,m+\alpha_c,n+\beta_c} &= Z_c^1 \cdot (1 - \Delta\alpha_c) \cdot (1 - \Delta\beta_c) + Z_c^3 \cdot \Delta\alpha_c \cdot (1 - \Delta\beta_c) \\ &\quad + Z_c^2 \cdot (1 - \Delta\alpha_c) \cdot \Delta\beta_c + Z_c^4 \cdot \Delta\alpha_c \cdot \Delta\beta_c, \\ \Delta\alpha_c &= \alpha_c - \lfloor \alpha_c \rfloor, \Delta\beta_c = \beta_c - \lfloor \beta_c \rfloor,\end{aligned}$$

$$Z_c^1 = x_{c,m+[\alpha_c],n+[\beta_c]}, Z_c^2 = x_{c,m+[\alpha_c],n+[\beta_c]+1},$$

$$Z_c^3 = x_{c,m+[\alpha_c]+1,n+[\beta_c]}, Z_c^4 = x_{c,m+[\alpha_c]+1,n+[\beta_c]+1}.$$



| | | |
|----|-----|-----|
| 23 | 82 | 20 |
| 44 | 91 | 210 |
| 32 | 162 | 95 |

| | | | | | |
|------------------|-----------------|--------------------------------------|-------|-------|-------|
| $\Delta\alpha_c$ | $\Delta\beta_c$ | $\tilde{x}_{c,m+\alpha_c,n+\beta_c}$ | 91 | 150.5 | 210 |
| 0 | 0.5 | 150.5 | 126.5 | 139.5 | 152.5 |
| 1 | 0.5 | 128.5 | | | |
| 0.5 | 1 | 152.5 | | | |
| 0.5 | 0 | 126.5 | 162 | 128.5 | 95 |
| 0.5 | 0.5 | 139.5 | | | |

[그림 3-20] Active Shift operation의 shift 값이 정수가 아닌 경우 bilinear interpolation

- shiftnet 중 Sparse Shift Layer (SSL)은 Core-Shift 만 남기면서 연산을 줄임과 동시에 우수한 성능을 확인함. Active Shift Net과 마찬가지로 미분 가능하며 추가적으로 양자화 기반 학습 방법임. [Chen, 2019a]
- 성능에 영향을 미치지 않는 메모리 Shift 연산을 가중하는 여분의 Shift를 제거하기 위한 방법으로 shift값 α, β 에 L1 정칙화(regularization)를 적용함

$$\mathcal{L}_{total} = \sum_{(x,y)} \mathcal{L}(f(x | W, \alpha, \beta), y) + \lambda \mathcal{R}(\alpha, \beta)$$

$$\mathcal{R}(\alpha, \beta) = \| \alpha \|_1 + \| \beta \|_1$$

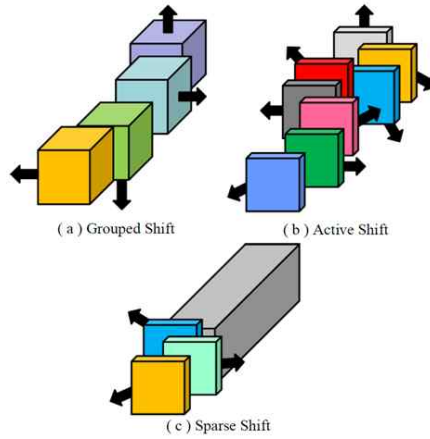
- Sparse ShiftNet에서는 미분 가능한 연산방법으로 Active Shift에서는 정수 값이 되지 않는 α, β 에 대해 보간 연산을 적용하는 대신, 정수 값이 되지 않는 α, β 에 대해 추정 정수 값으로 대체하는 연산을 적용함

- 순전파 연산

$$O_{c,i,j} = I_{c,i+|\alpha_c|_+,j+|\beta_c|_+}$$

| • | † 연산은 반올림 연산임

- 역전파 연산



[그림 3-21] Group Shift, Active Shift 그리고 Sparse Shift

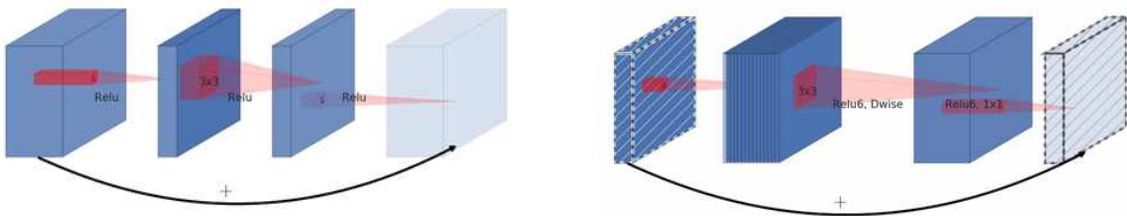
연산 수식 $\frac{\partial L}{\partial I_{c,i,j}} = \frac{\partial L}{\partial O_{c,i-|\alpha_c|_t, j-|\beta_c|_t}}$ 을 시작으로,

shift 값 α_c, β_c 각각에 대해 편미분 한 결과,

$$\frac{\partial L}{\partial \alpha_c} = \sum_i^w \sum_j^h \frac{\partial L}{\partial O_{c,i,j}} \sum_{(n,m) \in \Omega} I_{c,n,m} \cdot (1 - |j + \beta_c - m|) \cdot \text{Sign}(n - i - \alpha_c)$$

$$\frac{\partial L}{\partial \beta_c} = \sum_i^w \sum_j^h \frac{\partial L}{\partial O_{c,i,j}} \sum_{(n,m) \in \Omega} I_{c,n,m} \cdot (1 - |i + \alpha_c - n|) \cdot \text{Sign}(m - j - \beta_c)$$

- Sparse ShiftNet에서는 MobileNet2에서 활용한 전통적인 잔류 블록이 아닌 역 잔류 블록(Inverted Residual Block)을 차용함. 역 잔류 블록은 전통적인 잔류 블록들은 채널수가 많은 계층들을 동일연결(identity connect)하는 반면, 역 잔류 블록에서는 채널수가 적은 계층들을 동일 연결함. 역잔류 블록은 기존 잔류



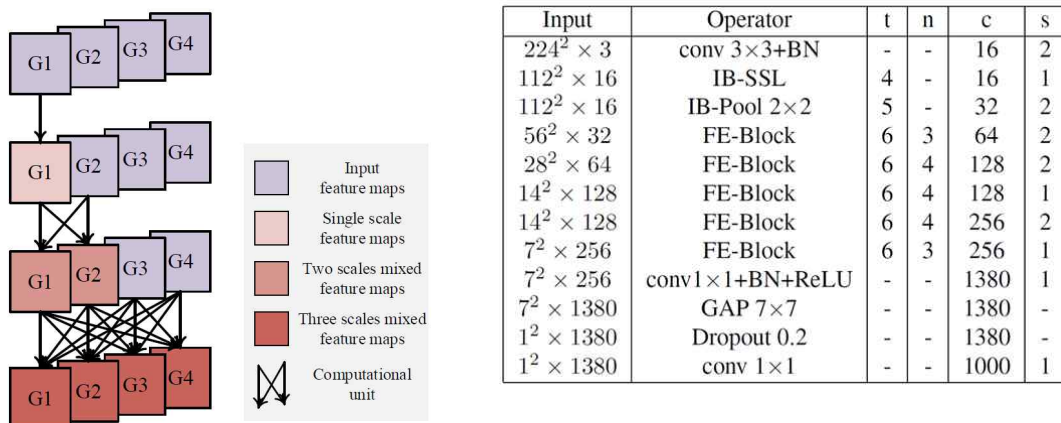
블록에 비해 정확도 성능은 향상되면서 메모리 사용량이 줄어듦.

(a) Residual Block

(b) inverted Residual Block[Sandler, 2018a]

[그림 3-22] Residual Block 과 Inverted Residual Block

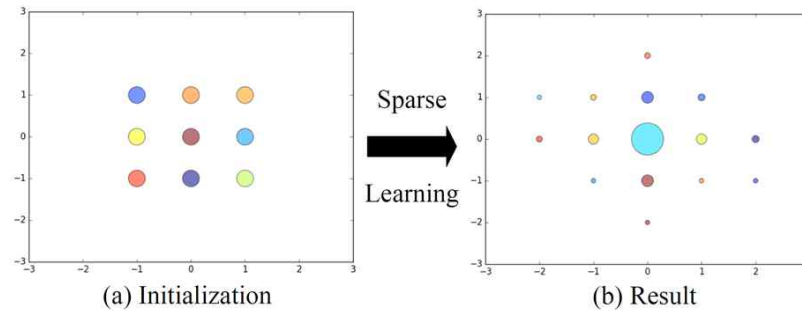
- Sparse ShiftNet은 FE-NET이라는 새로운 뉴럴 네트워크 구조를 제안함. Fe-NET은 특징 맵이 컨벌루션 연산이 반복 되면서 작은 부분 집합들로 붕괴되어 가는 특징이 있다는 분석의 내용을 기반으로 함(Zhang, 2016).
 - 특징 맵을 채널 방향으로 적제(channel wise concatenation)하되, 연산 블록들을 적제 할 때, (1) 기본 연산블록은 역병목 블록을 기본 단위로 하고,
 - (2) 채널 방향 분리와 채널 방향 적제연산을 통해 일부 하나의 블록을 4개의 계층로 구성하되 채널의 개수를 점차적으로 늘리며 적제 하는 구조를 취함.
 - 이는 densenet과 유사한 구조를 취하나 densenet에서는 특징 맵을 적제 하여서 개수를 늘렸다가 변환 계층(transition layer)을 줄이는 형태의 연산을 반복하였으나, 제안 FE Block은 특징 블록들을 채널 방향 적제연산을 통해 inception 모듈에서 특징 맵을 혼합하여 적제 한 것과 유사한 구조를 지님. (그림 3-23참조)
- Sparse 학습을 통해 shift가 취해지지 않은 비율로 shift가 더욱 중요한 역할을 하는 계층과 그렇지 않은 계층이 있음을 확인 (그림 3-23b). Shift가 가장 많은 비율을 차지한 Shift-ResNet20의 블록2_2 (전체 채널 맵중 24%가 shift 특징 맵으로 구성)의 sparse 학습 전후 특징 맵들의 중심 지점의 구성을 통해 shift 위치가 일정한 비율로 학습을 시작하여서 그 위치가 다양하게 흩어진 모양을 확인할 수 있음. (그림 3-23) [Chen2019a]



(a) FE-Block Network 구조 (b) Fe-Block의 network구조, t는 확장 비율, n은 FeBlock개수, c는 출력 채널수를 나타내며, s는 stride를 표기함

[그림 3-23] FE-Block의 구조와 FE-BLOCK을 이용한 Shift Network의 구조

- [그림 3-24]에서 영역의 각 지점은 같은 이동 패턴(shift pattern)을 보이는 특징 맵 내의 채널의 개수를 나타냄. x축과 y축은 좌우와 상하의 위치를 각각 나타냄.



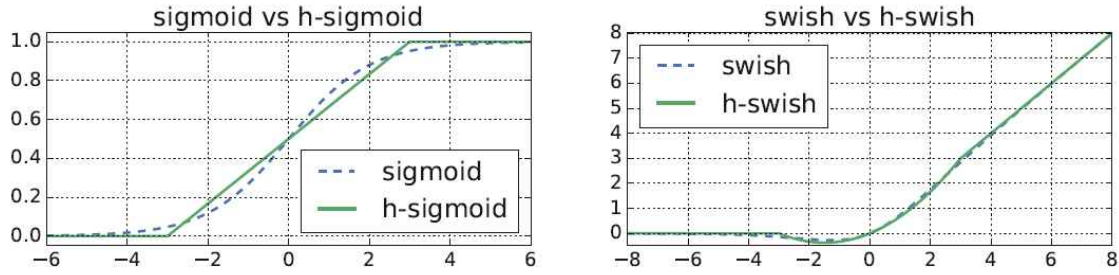
[그림 3-24] Sparse 학습 후 ResNet20의 블록2_2 이동 위치 분포
76% shift 맵으로 구성된 ShiftResNet20의 Block2_2

8. Searching for MobileNet V3 (Howard, 2019a)

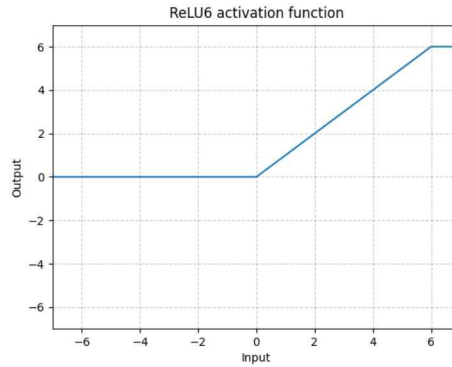
- 기존의 MobileNet V1, MobileNet V2에서 성능 향상에 도움이 된 디자인을 차용하여 사용함. MobileNet V2에 사용된 역 잔류 블록과 선형 병목 구조를 차용하였는데, 역잔류블록은 앞서 7. ShiftNet에서 설명됨.
- 선형 병목구조란 RELU함수가 비선형 매핑 (0이하의 값을 모두 0에 매핑)구조로 인한 정보 손실을 보완하고자 함. 저차원의 입력 데이터를 저차원 공간에 매핑하는 것 보다 고차원의 공간에 매핑하는 것이 데이터 손실이 적음을 착안하여, 입력 데이터에 확장 층(expansion layer)을 추가하여 그 차원을 증가시킨 후 depth-wise 컨벌루션을 하고 최후에 투사(projection)층을 통해 데이터를 보존하는 형태로 설계됨. 이는 채널수를 늘려 그 정확도를 높여가는 방법이므로 연산 성능 및 메모리 사용 측면에서는 이득이 없는 연산으로 분석됨.
- 정확도 향상을 위해 RELU를 대체하는 함수로 swish 함수를 제안 사용함

$$swish(x) = x \cdot \sigma(x) \quad (8-1)$$

이 연산은 정확도는 향상시키나 sigmoid연산이 RELU연산에 비해 연산량 증가를 가져오므로 연산 속도 저하를 막고자 hard sigmoid 함수와 부분 선형 hard analog로 대체하여 hard swish 함수를 사용함.



[그림 3-25] Sigmoid, hard-sigmoid, Swish, hard-Swish 함수

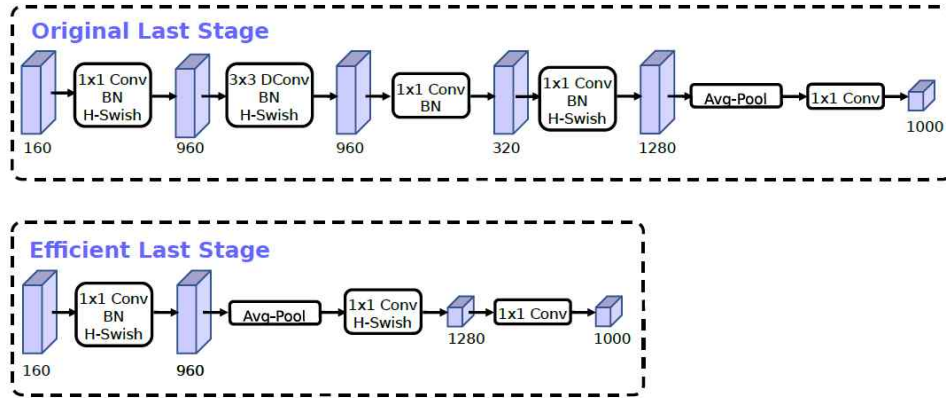


[그림 3-26] RELU6 함수

$$h-swish[x] = x \frac{ReLU6(x+3)}{6} \quad (8-2)$$

- Sigmoid, Swish를 사용한 경우와 hard-sigmoid, hard-swish 함수를 사용하였을 경우 정확도 성능에는 차이가 없으나 연산측면에서 이득이 있음. 즉 h-swish함수는 간단한 함수로 구현 가능하므로 지연 비용(latency cost)을 줄여 메모리 접근 횟수를 줄이는 이득이 있음
- 최적의 뉴럴 네트워크 구조를 찾는 데 있어 NAS방법을 사용하여 최적화된 블록 구조를 찾고 각 계층별 최적화된 필터를 찾는 형태로 계산됨
- 최적의 뉴럴 네트워크 구조를 찾은 후 연산비용이 비싼 계층들에 대해 정확도는 유지하면서 지연시간(latency)을 줄이기 위한 수정을 진행함
- 뉴럴 네트워크 계층 마지막 단계에 mobilenet2를 통해 제안된 역 잔류 블록 구조가 적용되어 있는데 이 마지막 단계가 지연시간을 크게 증가시키는 원인 블록으로 확인. 지연 시간을 줄이기 위해 역 잔류 블록에서 확장 하는 부분을 남기고 평균 통합(average pooling)을 먼저 수행한 후 이후에 1x1 컨벌루션을

수행하는 구조로 대체함. 해당 뉴럴 네트워크 구성을 통해 정확도는 유지되면서 연산성능은 지연 시간을 7ms줄이고 실행시간 연산 성능의 11%를 향상시켰는데 이는 30M MAdds 연산을 줄이는 것과 같은 효과임.



[그림 3-27] 마지막 뉴럴 네트워크 계층의 수정, 이를 통해 지연시간 성능 11% 향상

- 뉴럴 네트워크의 초기 필터 집합의 필터의 개수는 3x3 컨벌루션으로 32개의 필터들을 보통 가지고 있는데 이러한 필터들에는 서로 다른 거울 이미지 (mirror image)들이 포함되어 있는 것을 확인함. 기존과 다른 비선형성을 더 하기 위해 hard-swish 함수를 사용해본 결과 16개의 필터로 필터개수를 줄여도 같은 정확도에 연산 지연시간을 2ms를 추가적으로 줄일 수 있었음.

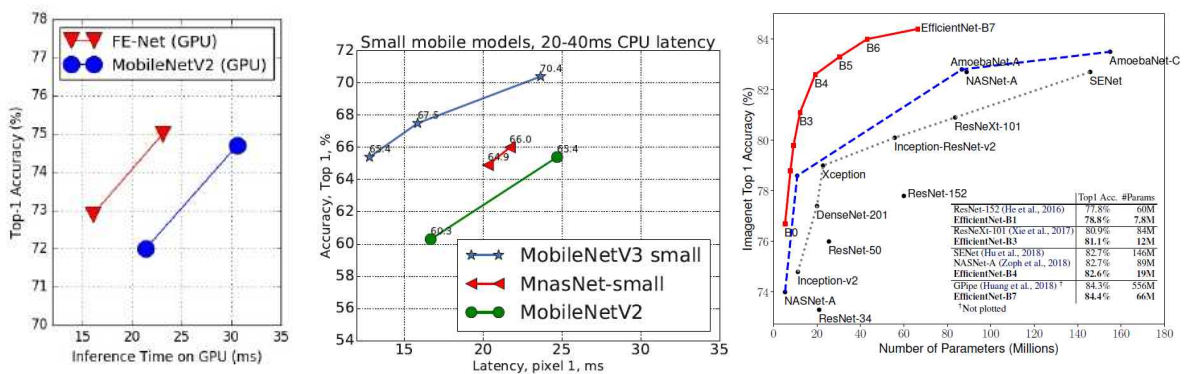
제4장

모델 경량화 연구 분야에 대한 분석

제 1 절 모델 경량화 기술의 평가를 위한 척도

1. 딥러닝 설계의 효율성을 측정하는 평가 척도

- 2012년 AlexNet 등장 이후 2019년에 이르기까지 최적의 모델을 찾기 위한 뉴럴 네트워크의 아키텍처는 연산량, 학습 파라미터 수를 줄이면서 성능을 향상시키는 것에 초점을 맞추어 왔음. 그러나 하드웨어 구조 설계의 효율성을 위해서는 연산량, 학습 파라미터 수, 정확도 뿐 만 아니라, 에너지 효율, 메모리 접근 시간을 줄이도록 고려한 설계가 보다 요구됨
- 보통 논문에서 딥러닝 설계의 효율성을 평가하는 척도로 가장 많이 등장하는 것은 그래프적인 해석으로 (1) 전체 학습 파라미터수 대비 정확도, (2) FLOPS 대비 정확도 (3) 추론 시간 대비 정확도 그리고 최근 (4) 지연시간 대비 정확도가 있음. [그림 4-1]에서 보는 것과 같이 파라미터 수 및 Flops가 덜 증가한 상태에서 더 높은 정확도를 보이는 좌상단으로 이어지는 그래프가 높은 성능을



드러냄

[그림 4-1] 효율성 평가 척도 왼편부터 (1) 추론 시간 대비 정확도 (shiftNet), (2) 지연시간 대비 정확도 (mobilenet-v3), (3) 학습 파라미터 수 대비 정확도(EfficientNet)

- 최근 딥러닝 설계의 효율성을 측정하는 평가 방법(metric)으로 (1) 학습 파라미터 수 대비 정확도의 비율을 측정하는 Information Density와 (2) 학습 파라미터 수 및 추론 시간동안의 연산 수 대비 정확도를 측정하는 NetScore가 있음.

■ Information Density $D(N)$

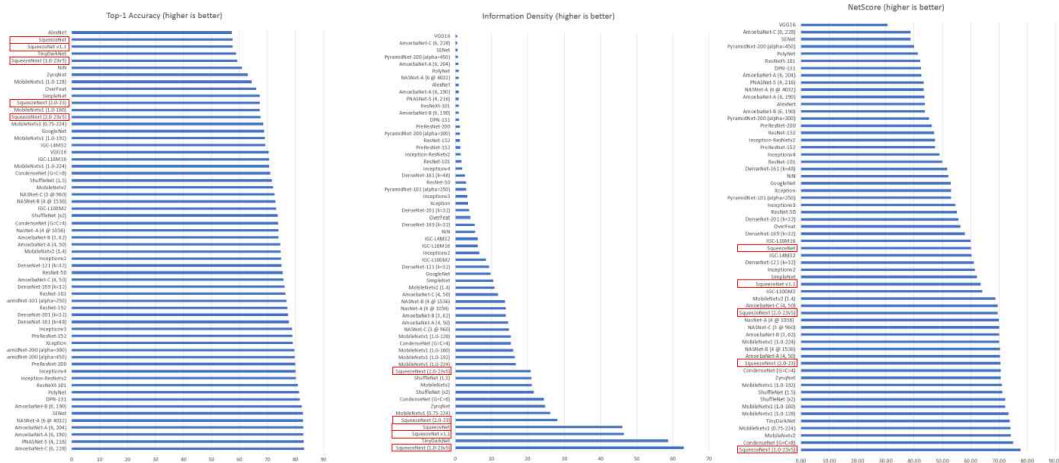
$$D(N) = \frac{a(N)}{p(N)}, \quad a(N) \text{은 정확도, } p(N) \text{은 학습 파라미터 수}$$

■ NetScore $\Omega(N)$

$$\Omega(N) = \log \left(\frac{a(N)^\alpha}{p(N)^\beta m(N)^\gamma} \right)$$

$m(N)$ 은 추론 시간 동안의 연산 수 (multiply-accumulate(MAC))

$$\alpha = 2, \beta = 0.5, \gamma = 0.5$$



[그림 4-2] (좌) Top1 정확도, (중) Information Density, (우) NetScore, 빨간색 - SqueezeNet

[Iandola, 2016a] 및 SqueezeNext [Gholami, 2018a] 모델 계열

- [그림 4-2]를 통해 확인할 수 있듯이 Information Density 및 NetScore 평가방법을 기준으로 SqueezeNet계열이 최적의 성능을 보이고 있음. 그러나 SqueezeNet은 information density와 netscore를 기준으로 매우 높은 점수를 얻었으나, 에너지 효율 측면에서는 Alexnet대비 1.3×10^8 배의 에너지를 요구함
- 연산 에너지를 가장 많이 요구하는 부분은 DRAM에서 ALU로 연산을 읽어 오는데 메모리 접근 부분임.

2. 딥러닝 모델 경량화 설계의 효율성을 측정하는 평가 척도

- 효율적인 디자인으로 설계된 Network에 대해 이를 양자화하여 학습 parameter 수를 줄이고 정확도를 높이기 위한 다양한 연구 결과물들이 있음. 경량화 연구의 취지에 맞춘 평가 척도가 이용되어야 하나 정확도를 높이기 위해 연산량을 대량으로 늘린다든지 학습 시간을 늘려서 전체 효율성을 낮추는 접근방법을 선별해 낼 필요가 있음. 이에 본 보고서의 저자는 5가지 metric을 기준으로 딥러닝 경량화 모델의 성능을 평가하여야 공정한 비교가 된다고 사료됨. 5가지 평가 방법 (1) 메모리 foot-print 크기, (2) 학습 시간, (3) 추론 시간, (4) 학습 중 최대 메모리 요구량, (5) 정확도를 기준으로 비교되어야 함.
- 앞서 언급된 Information Density와 NetScore는 학습 파라미터 개수와 정확도 그리고 추론시간까지 고려된 평가 방법이나 해당 평가 방법에서는 학습을 위해 필요한 자원 요구량이 되는 학습 시간과 학습 시간 중 최대 메모리 요구량에 대해 고려되지 않은 제약사항이 따름.

제 2절 절 현재 모델 경량화 기술의 한계

- 5가지 평가 방법 (연산량, 파라미터 수, 정확도 뿐만 아니라, 에너지 효율, 메모리 접근 속도)을 모두 고려한 효율적인 네트워크 설계를 위해서는 입력 채널의 수와 출력 채널의 수가 일치할 때 MAC(메모리 접근 시간) 수가 줄어들므로 일치시켜야 함

- h, w 를 특징 맵의 크기라 하면, 1 컨벌루션의 FLOP B 는

$$B = hwC_1C_2$$

- 메모리 접근 비용 (MAC), 즉 메모리 접근 연산의 수는

$$MAC = hw(C_1 + C_2) + C_1C_2$$

- MAC은 FLOP에 의해 주어지는 최저 경계 값을 갖는데, 입력과 채널수가 일치할 때, 최저 경계 값(lower bound)에 이름.

$$MAC \geq \sqrt[3]{hwB} + \frac{B}{hw}$$

- 과도한 그룹별 컨벌루션은 MAC연산을 증가 시키므로 그룹수를 줄여야 함

- 1x1 그룹별 컨벌루션에 대한 MAC과 FLOPS의 관계는 그룹의 수를 g 라 할 때 다음과 같음.

$$\begin{aligned} MAC &= hw(c_1 + c_2) + \frac{c_1 c_2}{g} \\ &= hwc_1 + \frac{Bg}{c_1} + \frac{B}{hw} \end{aligned}$$

- 고정된 h, w, c_1 이 주어졌을 때 연산 비용 B 와 MAC는 그룹 g 의 증가에 따라 증가함.

○ 네트워크 조각별로 나누는 연산 방법은 병렬연산 정도를 낮추므로 줄여야 함

- googleNet에 많이 활용된 자동 생성 아키텍처인 여러 갈래(multi-path) 구조는 다양한 네트워크 블록에 차용됨[Szegedy, 2014a]. 이는 큰 연산 대신 많은 연산을 쪼개어 수행하는 구조를 띄고 있음. 이러한 구조는 정확도 향상에는 도움이 되나, GPU와 같은 강력한 병렬 연산이 가능한 구조에서는 부적절한 연산이며, kernel 띄우거나 synchronization과 같은 추가 오버헤드(overheads)를 가져오게 됨

○ 엘리먼트 별(Element-wise) 연산도 많은 연산량을 차지하므로 줄여야 함

- RELU, AddTensor, AddBias 등의 연산은 엘리먼트 별 연산자들로 작은 FLOPS를 갖으나 상대적으로 큰 MAC연산을 수행함.

○ 양자화 연구의 공정한 평가가 진행되어야 함. 경량화 연구의 취지에 맞춘 평가 척도가 이용되어야 하나 정확도를 높이기 위해 연산량을 대량으로 늘린다던지 학습 시간을 늘려서 전체 효율성을 낮추는 접근방법을 선별해 낼 필요가 있음. 이에 본 보고서의 저자는 5가지 metric을 기준으로 딥러닝 경량화 모델의 성능을 평가하여야 공정한 비교가 된다고 사료됨. 5가지 평가 방법 (1) 메모리 내 학습 파라미터 저장 크기, (2) 학습 시간, (3) 추론 시간, (4) 학습 중 최대 메모리 사용 요구량, (5) 정확도를 기준으로 비교되어야 함.

제 3 절 적절한 모델 경량화 기술의 선택 방안

- 최근 ImageNet 데이터셋의 ILSVRC SOTA 모델들이 된 Efficient Net 및 RegNet을 통해 데이터 셋의 크기와 시스템 가용 리소스를 고려한 효율적인 모델에 관한 제안이 있어 왔음. AutoML을 통해 선택된 ImageNet 계열의 SOTA 모델을 이용하여서 시스템 가용 리소스에 맞춰 적절한 모델 경량화 기술이 선택 가능하도록 구사되어 옴
- EfficientNet의 기본 모델로 사용된 mNAS net을 통해 AutoML로 얻어낸 EfficientNet-B0 기본 뉴럴 네트워크 구조는 입력 이미지 해상도 224x224 전체 채널의 개수 2096, 컨벌루션 총 계층의 개수는 18 이며, 계층별 해상도와 채널 수 그리고 각 블록별 계층들의 개수는 <표 4-1>과 같음.

<표 4-1> EfficientNet의 Baseline model (mNASNet으로 구해진 최적화 모델)

| Stage i | Operator Fi | Resolution Hi×Wi | Channels Ci | Layers Li |
|---------|------------------------|------------------|-------------|-----------|
| 1 | Conv 3x3 | 224×224 | 32 | 1 |
| 2 | MBConv1, k3x3 | 112×112 | 16 | 1 |
| 3 | MBConv6, k3x3 | 112×112 | 24 | 2 |
| 4 | MBConv6, k5x5 | 56×56 | 40 | 2 |
| 5 | MBConv6, k3x3 | 28×28 | 80 | 3 |
| 6 | MBConv6, k5x5 | 14×14 | 112 | 3 |
| 7 | MBConv6, k5x5 | 14×14 | 192 | 4 |
| 8 | MBConv6, k3x3 | 7×7 | 320 | 1 |
| 9 | Conv1x1 & Pooling & FC | 7×7 | 1280 | 1 |

- Efficient Net 에서는 (깊이, 너비, 해상도) = (학습 계층 개수 α^ϕ , 채널 개수 β^ϕ , 입력 이미지의 해상도 γ^ϕ)의 크기 증가에 따라 요구 연산량 (FLOPS 및 학습 파라미터수)와 정확도 사이의 균형에 따라 8단계의 모델(EfficientNetB0에서 EfficientNetB7까지)을 제시함. EfficientNet-B1의 학습 파라미터개수가 7.8M³⁾이면서 Top1 정확도가 78.8%인 반면, EfficientNet-B7의 학습 파라미터수는 66M 이면서 정확도가 84.4%에 이름. 즉 시스템 요구 리소스가 달라짐을 가늠하는데 사용가능함 (그림 3-19 참조).
- EfficientNet에서 compound 확장기법으로 실험에 이용하고자 하는 시스템의 메모리와 flops 제한에 따라 정확도를 최대한으로 늘리는 조건을 찾고 자 하는 디

3) M은 Million 단위 임

$$\begin{aligned}
& \max_{d,w,r} \text{Accuracy}(\mathcal{N}(d,w,r)) \\
& s.t. \quad \mathcal{N}(d,w,r) = \bigodot_{i=1 \dots s} \hat{\mathcal{F}}_i^{d \cdot \hat{L}_i}(X_{\langle r \cdot \hat{H}_i, r \cdot \hat{W}_i, w \cdot \hat{C}_i \rangle}) \\
& \quad \text{Memory}(\mathcal{N}) \leq \text{target_memory} \\
& \quad \text{FLOPS}(\mathcal{N}) \leq \text{target_flops}
\end{aligned}$$

자인에 기반함. (깊이, 너비, 해상도) = $(\alpha^\phi, \beta^\phi, \gamma^\phi)$ 을 2배씩 늘려 가는데 $(\alpha \cdot \beta^2 \cdot \gamma^2)^\phi \approx 2$ 있어 최적의 α, β, γ 값을 구하고자 함 (Tan 2018a).

$\alpha = 1.2, \beta = 1.1, \gamma = 1.15$ 가 최적의 조건으로 드러났으며 B1-B7을 ϕ 을 늘려가며 찾음.

- RegNet은 모델의 최적화된 디자인 공간을 찾는 방법에 여러 비율 대비 정확한 성능을 드러내는 모델의 비율을 확인하는 방법으로, 랜덤 뉴럴 네트워크 연결을 통해 뉴럴 네트워크 구성을 하던 기존 논문(Xie, 2019)의 확장 선상의 논문으로 볼 수 있음. 그런데 사용 뉴럴 네트워크의 리소스가 매우 큰 대용량의 시스템 공간에서의 모델들만을 찾은 한계가 었보임(Radosavovic, 2020)
- EfficientNet 이나 RegNet이 실험 대상으로 삼은 ImageNet 데이터는 현실 세계에서 실험이 필요한 데이터 셋의 크기가 항상 ImageNet과 같은 대용량 크기의 데이터셋은 아니므로 실험을 통해 얻고자 하는 데이터셋의 이미지 크기를 고려한 기본 모델 선택이 되어야 함.
- 또한 경량화를 수행한 목적에 맞는 경량화 모델 연구가 진행되어야 함. 예를 들어, 모바일 디바이스에서 실시간 사용자들의 서비스를 목적으로 경량화 모델을 진행한 것이라면 학습시간이 오래 걸리거나 정확도 성능이 조금 떨어지더라도 메모리 foot-print 크기와 추론 시간이 주요 평가 방법이 되어 해당 부분을 집중한 경량화를 진행한 kill-the-bits와 같은 모델이 목적에 맞는 모델이 됨. 그러나 모델 경량화의 목적이 학습 시간 단축 및 시스템 제한 리소스를 고려한 것이라면, 모델의 주요 평가 방법은 학습 시간 단축과 실시간 시스템 자원 요구량 그리고 공정한 평가를 위한 정확도 향상이 목적이 되어야 함.

제5장

모델 경량화 기술의 활용방안과 전망

제 1 절 활용 방안과 기대효과

1. 활용 방안

- 딥러닝 모델 개발 및 활용에 있어서의 효율성 제고
 - 개발하는 모델의 크기 축소를 통한 학습 및 추론 과정의 시간, 비용 감소
 - 이를 통해 전체 딥러닝 모델 개발 및 운용 과정에서의 비용 감소 효과
- KISTI 3대 연구영역으로 확장
 - 콘텐츠 큐레이션, NDSL, NTIS, 공유 플랫폼 등의 지능형 데이터 처리에 활용
 - 슈퍼컴퓨팅, 네트워크, 대용량 데이터를 동시에 활용한 초고성능의 인공지능 응용 기술 개발에 적용
 - 데이터분석 플랫폼 개발, R&D 투자 분석 파이 시스템 등의 빅데이터 분석에 활용
- 국내·외 기술 보급
 - 국내·외 대학, 연구소, 기업 등으로 보급을 통해 데이터 분석의 경제성과 AI 결정의 신뢰성 확보를 도모
 - R&D 연구자, 개발자, 학생들의 데이터 및 AI 전문 교육을 통해 국가적 전문성 향상에 기여

2. 기대 효과

- 딥러닝 모델 경량화를 통한 모델 개발 및 운용 과정에서의 효율성 제고를 통한 비용 및 시간 감소 효과
- 데이터 자동 선별·확장 기술 개발을 통해 인공지능 데이터 학습 비용 및 시간 절감 또한 가능
- 기관 내의 인공지능 전문 인력 양성을 통한 다양한 R&D 영역 지원 가능

제 2 절 모델 경량화 기술 발전 전망

- 딥 러닝 모델이 갈수록 모델 크기와 연산량이 커감에 따라 학습과 추론 과정에서 요구되는 전산 자원과 그로 인한 비용이 급격히 증가
- 또한, 다양한 응용들이 딥 러닝 모델 기반으로 개발됨에 따라 모바일 기기 등에서의 제한된 정보 자원을 활용한 추론 지원 또한 요구된다. 이러한 분야에서 모델 경량화 기술은 지속적으로 큰 관심을 받을 것으로 기대한다.
- 모델 경량화 연구에 있어 하나의 문제점은 학계에서 발표된 코드들이 그 자체로 실현 가능하기보다는 가능성을 제시하기 위한 일종의 데모 형식으로 되어 있어 여러 평가 척도를 통한 비교가 곤란함
- 이는 대부분의 모델 개발에 사용되는 Python에서의 데이터 타입이 묵시적으로 선언되고, 하나의 객체로써 인식되어 실제 비트 연산이 C에서의 비트 연산이 아닌, H/W 적인 고려 없는 코드들이 테스트되고 있기 때문.
- 따라서, 모델 경량화의 정확한 평가를 위해서는 단일한 평가 척도와 실제 운영 H/W 환경과 부합되는 평가 체계가 마련되어야 할 것으로 보임.

Acknowledgement

- 이 보고서는 한국과학기술정보연구원의 주요사업 지능형인프라기술연구(K-20-L04-C01)의 결과물로 작성되었음.
- 이 보고서의 주요 내용들은 저자들의 논문인 [Lee2019b], [Kim2020a]의 내용을

바탕으로 하고 있으며, 2장 1절의 CNN 모델의 학습과정 개요의 경우 CHML 필명의 블로그 <https://untitledblog.tistory.com/150> 의 글과 그림을 많은 부분 인용하고 있음을 밝힘

참고 문헌

- [Abadi2016a] Abadi, Martín et al., Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467, 2016
- [Alemdar2017a] Alemdar, Hande, et al. "Ternary neural networks for resource-efficient AI applications." 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, 2017
- [Alex2012a] Alex Krizhevsky et al. "ImageNet Classification with Deep Convolutional Neural Networks," NIPS 2012
- [Bendersky2018] Eli Bendersky, "Depthwise Seperable Convolutions for machine learnig,"URL: <https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning/>, (2018)
- [Bianco2018a] Bianco, Simone, et al. "Benchmark analysis of representative deep neural network architectures." IEEE Access 6 (2018): 64270–64277.
- [Bengio2013a] Bengio et al., Estimating or propagating gradients through stochastic neurons for conditional computation. CoRR, abs/1308.3432, 2013
- [Ben-Nun2019] Ben-Nun, Tal, and Torsten Hoefler. "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis." ACM Computing Surveys (CSUR) 52.4 (2019): 1–43.
- [Brown2020a] Brown, Tom B., et al. "Language models are few-shot learners." arXiv preprint arXiv:2005.14165 (2020)
- [Chawla2002a] Chawla, N. V. et al., SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research 16:321–357, 2002
- [Chen2018] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, andHartwig Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," ECCV (2018)
- [Chen2019a] Weijie Chen et al. "All You Need is a Few Shifts: Designing Efficient Convolutional Neural Networks for Image Classification." The IEEE Conf on CVPR, pp. 7241–7250 (2019)
- [Chen2019b] Shangyu Chen, Wenya Wang, "MetaQuant: Learning to Quantize by Learning to Penetrate Non-differential Quantization," NIPS (2019)
- [Courbariaux2015a] Courbariaux et al., "Binaryconnect: Training deep neural networks with binary weights during propagations." Advances in neural information processing systems. 2015.

- [Courbariaux2016a] Courbariaux et al., “Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 and -1,” Proceedings of NIPS 2016.
- [Dauphin2015] Y. Dauphin, H. de Vries, J. Chung, and Y. Bengio, “RMSprop and equilibrated adaptive learning rates for non-convex optimization. Technical Report,” arXiv:1502.04390, (2015)
- [Doan2018a] Doan A.H. et al., Deep learning for entity matching: a design space exploration, In proceedings of ACM International Conference on Management of Data pp.19–34, 2018
- [Dong2019a] Dong, Zhen, et al. "Hawq: Hessian aware quantization of neural networks with mixed-precision." Proceedings of the IEEE International Conference on Computer Vision. 2019.
- [Fisher2018a] Fisher, A. et al., All Models are Wrong but many are Useful: Variable Importance for Black-Box, Proprietary, or Misspecified Prediction Models, using Model Class Reliance, arXiv preprint, arXiv:1801.01489, 2018
- [Fu2017] Gang Fu, Changjun Liu, Rong Zhou, Tao Sun, Qijian Zhang, "Classification for high resolution remote sensing imagery using a fully convolutional network," Remote Sensing, (2017)
- [Gholami2018a] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, Kurt Keutzer, “SqueezeNext: Hardware-Aware Neural Network Design,” CVPR (2018)
- [Gong2011] Y. Gong and S. Lazebnik, "Iterative Quantization: A Procrustean Approach to Learning Binary Codes," Proc. IEEE Conf. on CVPR (2011)
- [Goodfellow2016] Ian Goodfellow, Yoshua Bengio, Aaron Courville, “Deep Learning,” <http://www.deeplearningbook.org>, MIT Press, (2016)
- [Guyon2019a] Guyon, I. et al., Analysis of the AutoML Challenge Series“, Automated Machine Learning, pp.177, 2019
- [Han2015a] Han, Song, et al. "Learning both weights and connections for efficient neural network." Advances in neural information processing systems, 2015
- [Han2015b] Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149, 2015
- [Han2018a] Song Han, William J. Dally, “Bandwidth-Efficient Deep Learning from Compression to Acceleration”, 55th ACM/ESDA/IEEE Design Automation Convergence (DAC),

- https://indico.cern.ch/event/714134/contributions/2977940/attachments/1640902/2620339/Song_Han.compressed.pdf, Jung 2018
- [Hao2019a] Karen Hao, Training a single AI model can emit as much carbon as five cars in their lifetime. MIT Technology Review, <https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/>, June 6. 2019
- [He2016a] Kaming He et al., "Deep Residual Learning for Image Recognition," The IEEE Conf. on Computer Vision and Pattern Recognition(CVPR) pp.770–778 (2016)
- [He2016b] Kaming He et al., "Identity Mappings in Deep Residual Networks," European Conference on Computer Vision (ECCV) pp.630–645 (2016)
- [He2018a] He, Y. et al., AMC : AutoML for model compression and acceleration on mobile devices, In proceedings of the European Conference on Computer Vision (ECCV), pp. 784–800, 2018
- [Hinton2015] Geoffery Hinton, Oriol Vinyals, and Jeff Dean, "Distilling the knowledge in a neural network," arXiv preprint arXiv:1503.02531, 2015
- [Howard2017a] Andrew G. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," The IEEE Conf. on CVPR arXiv:1704.04861 (2017)
- [Howard2019a] Andrew Howard, M Sandler, G. Chu, L.C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, H. Adam, "Searching for MobileNetV3," ICCV 2019
- [Hu2018a] Hu, Jie, Li Shen, and Gang Sun. "Squeeze-and-excitation networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018
- [Huang2017a] Gao Huang et al. "Densely Connected Convolutional Networks." The IEEE Conf. on CVPR, pp.4700–4708 (2017)
- [Huang2019a] Huang, Yanping, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." Advances in neural information processing systems. 2019.
- [Iandola2017] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, Kurt Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size", ICLR 2017
- [Jeon2018] Yunho Jeon, Junmo Kim, "Constructing Fast Network through Deconstruction of Convolution," 32nd Conf. on NeurIPS Spotlight (2018)
- [Jiang2018a] Jiang, Jiawei, et al. "SketchML: Accelerating distributed machine learning with data sketches." Proceedings of the 2018 International Conference on Management of Data,

2018

- [Jouppi2017] Jouppi, Norman P., et al. "In-datacenter performance analysis of a tensor processing unit." Proceedings of the 44th Annual International Symposium on Computer Architecture. 2017
- [Kang2019a] Kang, U., Lightweight deep learning with model compression, Tutorial, In proceedings of 6th IEEE Int'l Conference on Big Data and Smart Computing, 2019.
- [Kim2017] Heehoon Kim, Hounghook Nam, Wookeun Jung, Jaejin Lee, "Performance Analysis of CNN Frameworks for GPUs," IEEE Intern. Symposium on Performance Analysis of Systems and Software(ISPASS) pp.55–64 (2017)
- [Kim2020] Eunhui Kim et al., "딥 러닝 모델의 경량화 기술 동향", Communications of KIISE 38(8):18–29, 2020
- [Krizhevsky2017a] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Communications of the ACM 60.6 (2017): 84–90
- [LeCun1989] LeCun, Yann, et al. "Backpropagation applied to handwritten zip code recognition." Neural computation 1.4 (1989): 541–551
- [LeCun1998] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278–2324
- [Lee2019] Hayun Lee, Donkyun Shin, "Performance and Energy Comparison of Different BLAS and Neural Network Libraries for Efficient Deep Learning Inference on ARM-based IoT Devices," Journal of KISSE, Vol. 46, No. 3, pp. 219–227, March (2019)
- [Lee2019b] Kyong-Ha Lee et al., "딥러닝을 위한 데이터 관리 기술 동향", Communications of KIISE 37(8):13–20, 2019
- [Li2014a] Li, Mu et al., Scaling distributed machine learning with the parameter server. In Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation(OSDI), 2014
- [Li2016a] Li, Hao, et al. "Pruning filters for efficient convnets." arXiv preprint arXiv:1608.08710, 2016
- [Liu2018a] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng, "Bi-Real Net: Enhancing the Performance of 1-bit CNNs With Improved Representational Capability and Advanced Training Algorithm," arXiv:1808.00278v5, 20 Sep. (2018)
- [Lundberg2017a] Lundberg, S. M. et al., A unified approach to interpreting model predictions, In Advances in Neural Information Processing Systems, pp. 4765–4774, 2017

- [Miao2017a] Miao, H. et al., Towards Unified Data and Lifecycle Management for Deep Learning, In Proceedings of 33rd IEEE ICDE, pp.571–582, 2017
- [Norouzi2013] Mohammad Norouzi, David J. Fleet, “Cartesian K-means,” IEEE Conf. on CVPR, pp.3017–3024 (2013)
- [Pham2018a] Hieu Pham et al. “Efficient Neural Architecture Search via Parameter Sharing.” arXiv:1802.03268 (2018)
- [Polino2018a] Polino, Antonio, Razvan Pascanu, and Dan Alistarh. "Model compression via distillation and quantization." arXiv preprint arXiv:1802.05668, 2018
- [Polyzotis2017a] Polyzotis, N. et al., Data management challenges in production machine learning. In Proceedings of the 2017 ACM International Conference on Management of Data, pp. 1723–1726. 2017
- [Radosavovic2020a] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaming He, Piotr Dollar, “Designing Network Design Spaces,” arXiv:2003.13678v1, 30 Mar (2020)
- [Rastegari2016a] Rastegari, M. et al. "XNOR-Net: Imagenet classification using binary convolutional neural networks." European conference on computer vision. Springer, Cham, 2016.
- [Ratner2017a] Ratner, A. et al., Snorkel: rapid training data creating with weak supervision, Proceedings of VLDB 11(3):269–282, 2017
- [Roh2018a] Roh, Y. et al., A survey on data collection for machine learning: a big data – AI integration perspective, arXiv:1811.03402v1, 2018
- [Sapunov2018a] Sapunov, Grigory, Hardware for Deep Learning. Part 3: GPU, <https://blog.inten.to/hardware-for-deep-learning-part-3-gpu-8906c1644664>
- [Sandler2018a] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov Liang-Chieh Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” CVPR (2018)
- [Silver2017a] Silver, D. et al., AlphaGo Zero: Learning from scratch, <https://deepmind.com/blog/alphago-zero-learning-scratch/>, 2017
- [Szegedy2015] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015
- [Stock2020] Pierre Stock, Armand Joulin, Remi Gribonval, Benjamin Graham, Herve Jegou, “And The Bit Goes Down: Revisiting the Quantization of Neural Networks,” Inter. Conf. on Learning Representation (ICLR) (2020)
- [Tan2018a] Mingxing Tan et al. “MnasNet: Platform-Aware Neural Architecture Search for Mobile.” The IEEE Conf. on CVPR (2018)

- [Tung2018a] Tung, Frederick, and Greg Mori. "Clip-q: Deep network compression learning by in-parallel pruning-quantization." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018
- [Waldrop2016a] Waldrop, M. Mitchell. "The chips are down for Moore's law." Nature News 530.7589 (2016): 144.
- [Wang2018a] Wang, Tongzhou, et al. "Dataset distillation." arXiv preprint arXiv:1811.10959, 2018
- [Wang2019a] Wang, Kuan, et al. "Haq: Hardware-aware automated quantization with mixed precision." Proceedings of the IEEE conference on computer vision and pattern recognition. 2019.
- [Wu2017] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzales, Kurt Keutzer, "Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions," The IEEE Conf. on CVPR (2017)
- [Xie2019] Saining Xie, Alexander Kirillov, Ross Girshick, Kaiming He, "Exploring Randomly Wired Neural Networks for Image Recognition," The IEEE Conf. on CVPR (2019)
- [Xin2019a] Xin, D. et al., Helix: accelerating human-in-the-loop machine learning. Proceedings of the VLDB Endowment, 11(12):1958–1961, 2019
- [Yang2019a] Yang, Zhilin, et al. "Xlnet: Generalized autoregressive pretraining for language understanding." Advances in neural information processing systems. 2019.
- [Yang2019b] Yang, Jiwei, et al. "Quantization networks." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019
- [Yin2020a] Yin, Hongxu, et al. "Dreaming to distill: Data-free knowledge transfer via DeepInversion." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020
- [Zhang2016] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun, "Accelerating Very Deep Convolutional Networks for Classification and Detection," IEEE Trs. on PAMI, Vol. 38, No. 10, Oc. (2016)
- [Zhang2018a] Zhang, Dongqing, et al. "LQ-nets: Learned quantization for highly accurate and compact deep neural networks." Proceedings of the European conference on computer vision (ECCV), 2018
- [Zhou2016a] S. Zhou, et al., "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," Proceedings of CVPR, 2016
- [Zoph2017a] Barret Zoph, Quoc V. Le. "Neural Architecture Search with Reinforcement

Learning.” arXiv:1611.01578 (2017)

[Zoph2017b] Zoph, Barret, et al. "Learning transferable architectures for scalable image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

딥러닝 모델 경량화 기술 분석

저자
이경하, 김은희

2020년 11월 26일 인쇄
2020년 11월 30일 발행

발 행 처



대전광역시 유성구 대학로 245

☎34141

전화 : 042-869-1004

등록 : 1991년 2월 12일 제 5-259호

발행인
최 희 윤

인쇄처
엠에스기획

