

# Deep Compression

(Compressing Deep Neural Networks with Pruning,  
Trained Quantization and Huffman Coding)

임세진

[https://youtu.be/miC\\_Tc3xw4c](https://youtu.be/miC_Tc3xw4c)

01. Deep Compression이 필요한 이유

02. Network Pruning

03. Trained Quantization and Weight Sharing

04. Huffman Encoding

05. 실험 결과

# 01. Deep Compression이 필요한 이유

- 모바일 환경에서의 딥러닝 기술

- 모바일 환경에서 딥러닝 기술을 사용하면 개인정보 보호, 실시간 처리, 네트워크 대역폭 면에서 **이점**이 있음
- 딥러닝 모델의 용량이 크기 때문에 모바일 어플에 딥러닝 모델을 그대로 탑재하기 **부담스러움**

【**Mobile** 환경 예시】

움직이는, 이동할 수 있는

→ 딥러닝 모델의 에너지 소비 문제



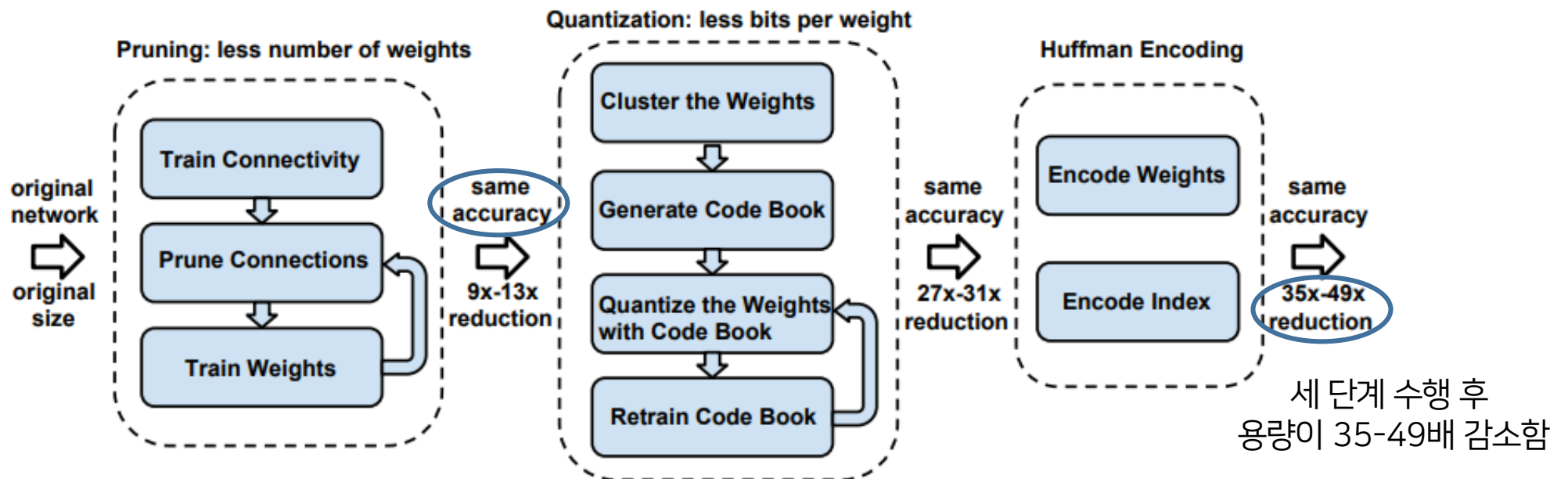
# 01. Deep Compression이 필요한 이유

- 딥러닝 모델의 에너지 소비 문제
  - 큰 딥러닝 네트워크를 사용할 때 가중치를 인출(fetch)하는 과정에서 많은 메모리 대역폭 요구
  - 포워딩을 위해 많은 내적 (dot product) 연산 수행
  - 메모리 접근 시 특히 많은 에너지 소비 발생

Sol → 뉴럴 네트워크의 용량 및 에너지 소비량을 줄일 수 있는 **Deep Compression**

# 01. Deep Compression이 필요한 이유

- Deep Compression 구성 : 3단계의 압축 파이프라인
  1. 가지치기 (Pruning) : 불필요한 연결은 숨겨 (가지치기하여) 중요한 연결만 남김
  2. 양자화 (Quantization) : 각 가중치를 나타내기 위한 bit 수를 감소시킴
  3. 허프만 인코딩 (Huffman Encoding) : 자주 등장하는 가중치에 작은 코드워드(codeword) 할당

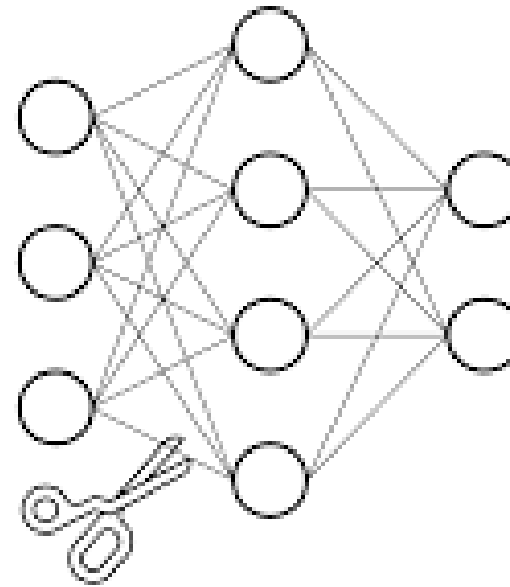
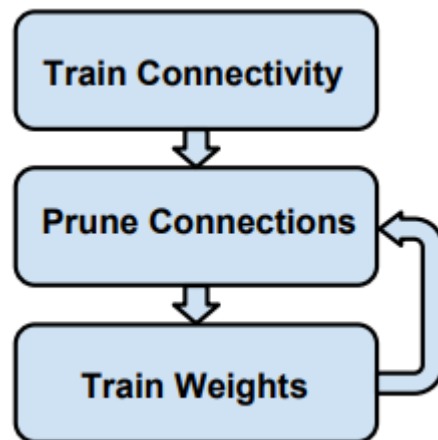


## 02. Network Pruning

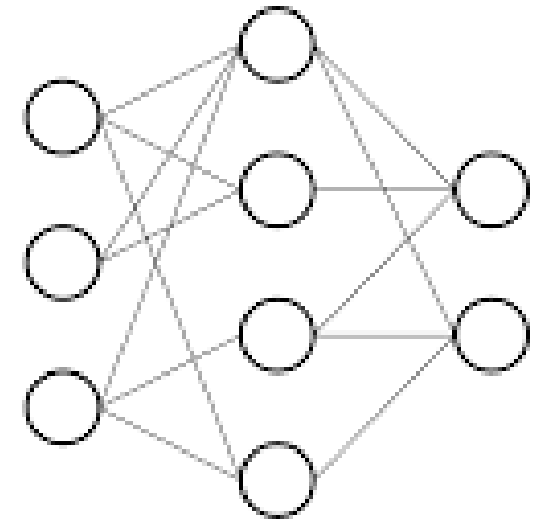
- 네트워크 가지치기

- 복잡도와 과적합 감소에 효과적 → 정확도 유지 / 증가

1. 일반적인 네트워크 학습 진행
2. 가중치 값이 작은 연결 제거 (절대값이 0에 가까운)
3. 남아있는 연결 유지 상태로 가중치 **재학습**



Before pruning



After pruning

## 02. Network Pruning

- 밀집행렬(Dense Matrix) / 희소행렬(Sparse Matrix)

- 가지치기 수행 결과 희소행렬 생성

Ex) 5X5의 가중치 행렬에서 가중치 값이 4 이상인 연결만 남기고 가지치기

1	2	1	0	3
3	2	1	6	2
5	0	2	1	1
0	4	1	2	6
0	1	3	2	7

원본행렬

가지치기



0	0	0	0	0
0	0	0	6	0
5	0	0	0	0
0	4	0	0	6
0	0	0	0	7

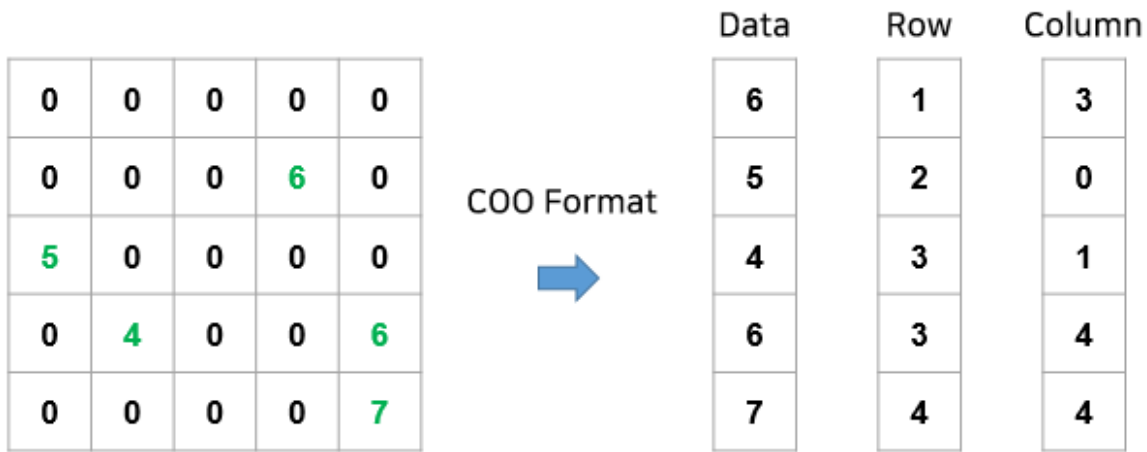
가지치기된 행렬

더 효율적인  
자료구조?

## 02. Network Pruning

- 희소구조 (Sparse Structure) :
  - Coordinate Format (COO)
  - Compressed Sparse Row (CSR) 일반적으로 CSR이 더 많이 메모리 절약

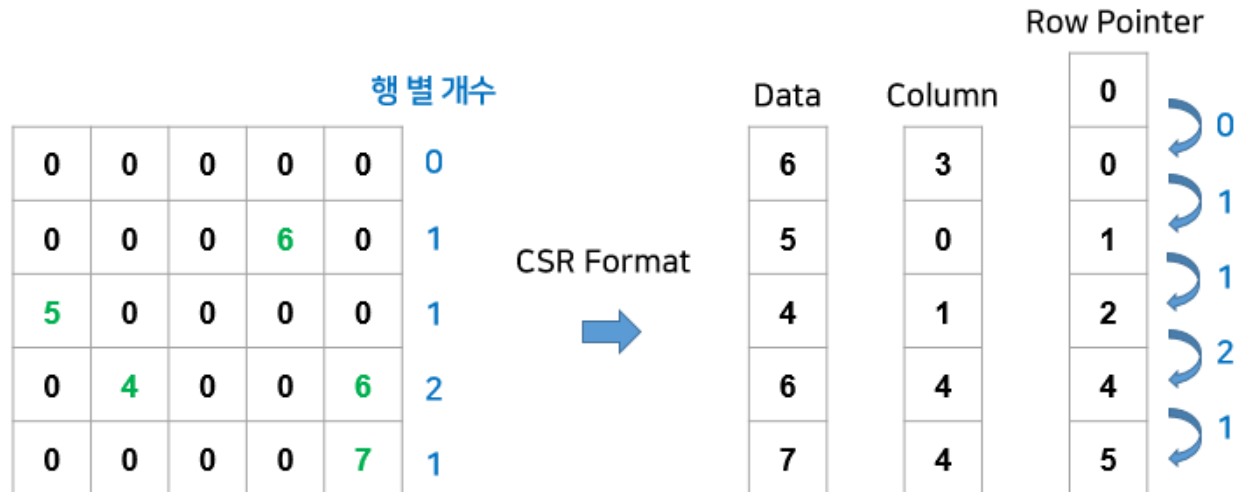
행렬에 포함된 0이 아닌 값만 행/열 위치 정보 기록



0이 아닌 원소의 개수가 a개일 때, 3a만큼의 공간이 요구됨

Coordinate Format

Row Pointer를 이용하여 표현



$2a + (n+1)$  만큼의 공간 요구됨

Compressed Sparse Row Format



## 02. Network Pruning

- Deep Compression에서는 희소구조를 저장하기위한 방법을 선택할 수 있음
  - ✓ CSR (Compressed Sparse Row)
  - ✓ CSC (Compressed Sparse Columns) : 열을 기준으로 압축하는 방식
- (  $2a+n+1$  ) 개의 수를 저장해야함

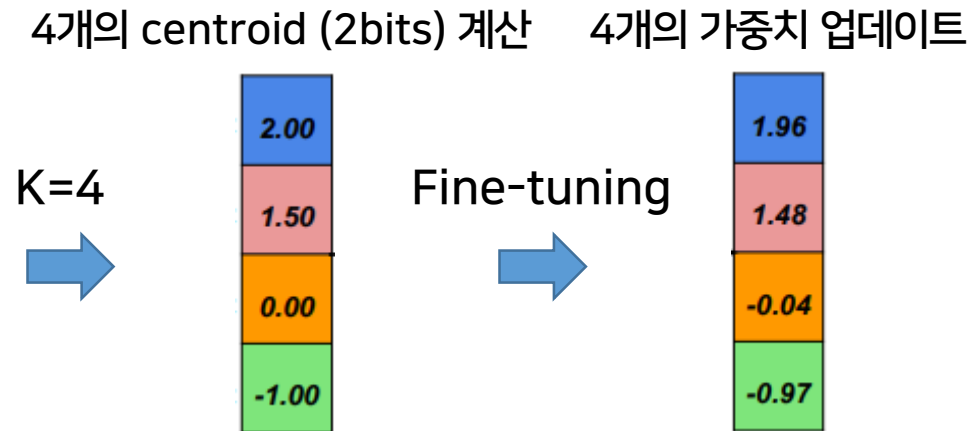
## 03. Trained Quantization and Weight Sharing

- 학습된 양자화와 가중치 공유
  - 양자화 : 각 가중치를 표현하기 위한 비트의 수를 감소시키는 효과가 있음
    1. 사용할 가중치의 개수 k를 설정
    2. K개의 가중치를 별도의 메모리에 저장한 후에 이를 공유
    3. K개의 가중치를 미세조정(fine-tuning)함 (정확도 최대화)

weights  
(32 bit float)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

원본 가중치 행렬



## 03. Trained Quantization and Weight Sharing

- 4 X 4 (n=16) 개의 가중치
- K = 4 개의 클러스터
- 각 연결당 비트 수 (b) = 32bits

압축률  
(Compression Rate)

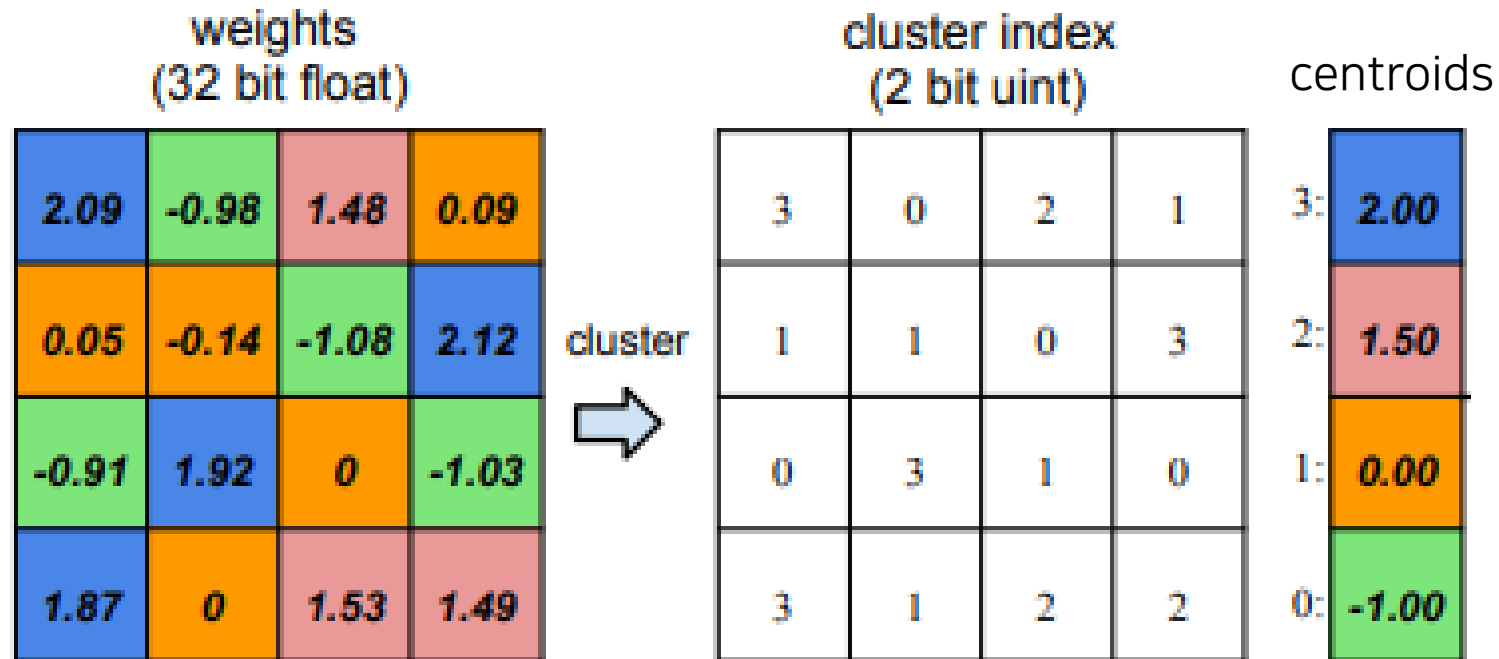


$$r = \frac{nb}{n \log_2(k) + kb}$$

$$3.2 = \frac{16 * 32}{16 * (2) + 4 * (32)}$$

압축 전  
압축 후

2bit      Floating point



## 03. Trained Quantization and Weight Sharing

- 가중치 공유 (Weight Sharing)를 위한 중심점(Centroid) 계산 : k-means clustering 알고리즘

Centroid = 공유 가중치 = 코드북(Codebook)

$$\arg \min_C \sum_{i=1}^k \sum_{w \in c_i} |w - c_i|^2$$

$C$  : K개의 클러스터 집합

$w$  : 학습데이터 역할을 하는 원본 가중치

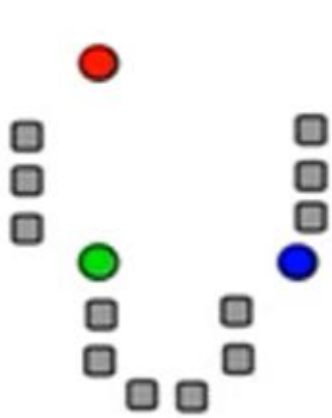
$w \in c_i$  : 각 클러스터에 포함되는 가중치

- 최솟값의 centroid를 업데이트하는 과정
- 원본 가중치와 유사한 값을 갖는 적절한 centroid를 찾을 수 있음

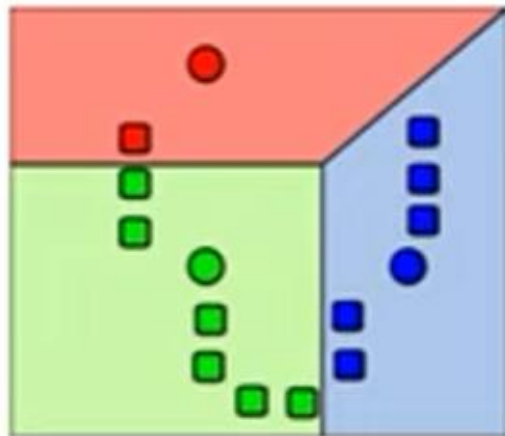
## 03. Trained Quantization and Weight Sharing

- K-means Clustering 알고리즘

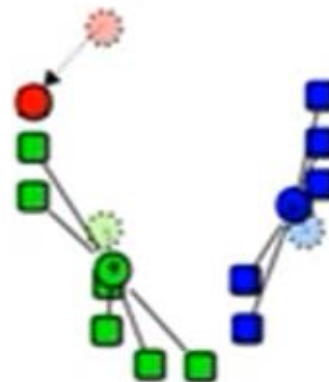
- 1) k개의 중심점을 초기화
- 2) 각 데이터는 가장 가까운 중심점을 기준으로 클러스터 구성
- 3) K개 클러스터의 중심점 값 조정
- 4) 중심점이 수렴할 때까지 2) 3) 과정 반복



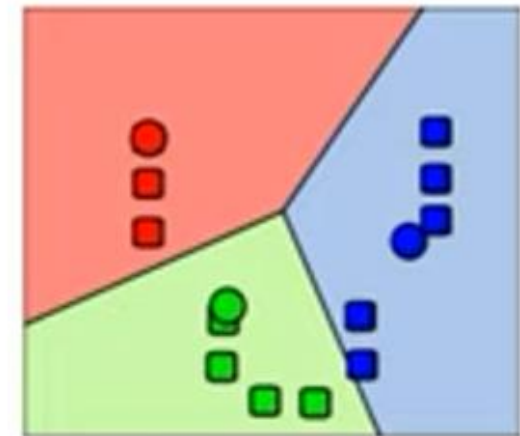
(1) 중심점 초기화



(2) 클러스터 구성



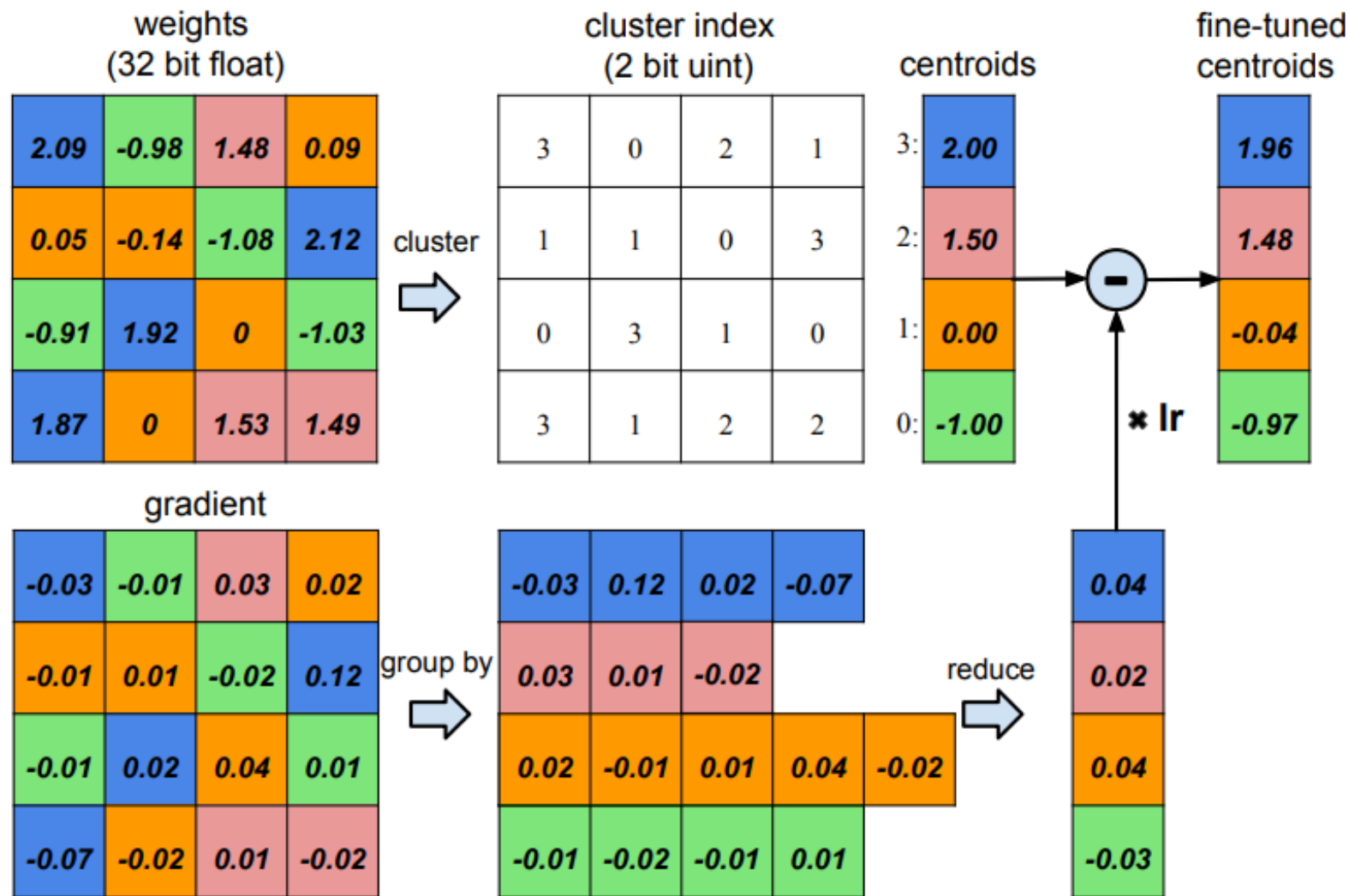
(3) 중심점 조정



업데이트 결과

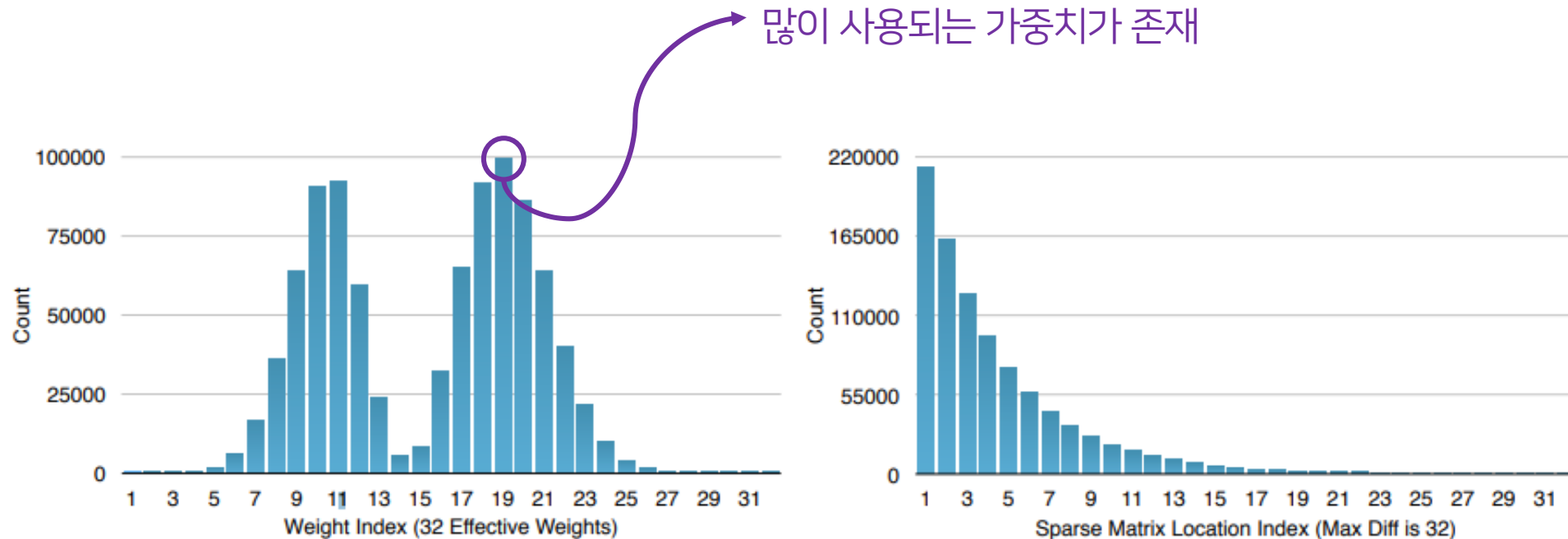
## 03. Trained Quantization and Weight Sharing

- centroid 갱신 → 보다 높은 정확도의 centroid 값 획득



## 04. Huffman Encoding

- 허프만 부호화 (무손실 데이터 압축 알고리즘)
- 특정한 중심점이 많이 등장 (biased distribution)
- 가변 길이 부호화 (Variable-length codewords) : 많이 등장하는 심볼에 적은 비트 할당



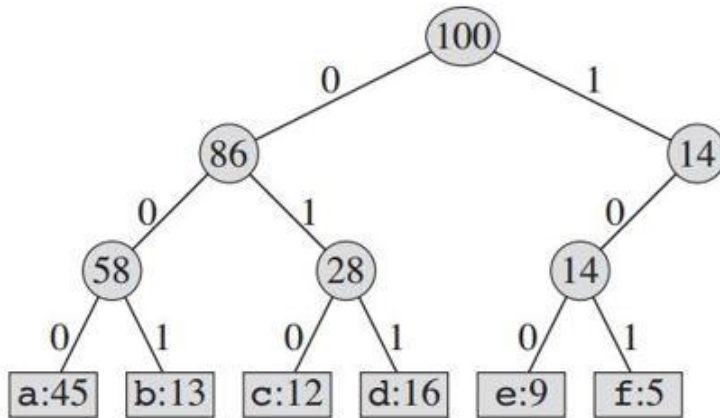
# 04. Huffman Encoding

- 허프만 부호화 (무손실 데이터 압축 알고리즘) *abc를 코드화*

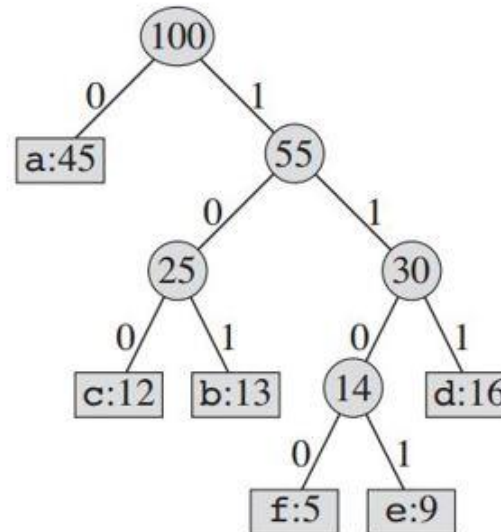
	a	b	c	d	e	f
fixed length codeword	000	001	010	011	100	101
variable length codeword	0	101	100	111	1101	1100
frequency (*1000)	45	13	12	16	9	5

000001010 (9 bits)

0101100 (7 bits)



Fixed-length Codeword




Variable-length Codeword



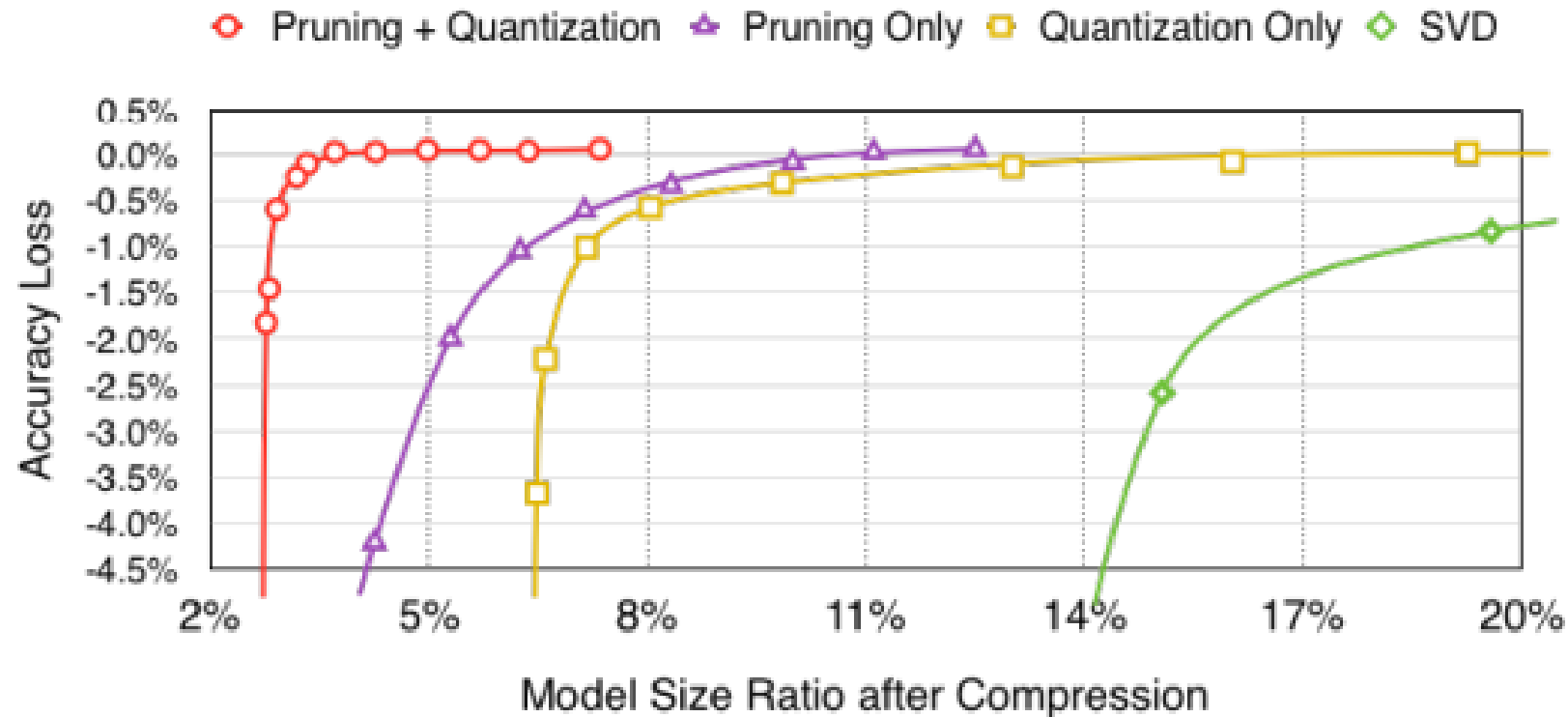
## 05. 실험결과

- Deep Compression을 사용하여 딥러닝 네트워크의 용량을 35배~49배까지 압축 가능
- 압축을 해도 정확도는 거의 그대로 유지 (성능 우수)

Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
LeNet-300-100 Ref	1.64%	-	1070 KB	
LeNet-300-100 Compressed	1.58%	-	<b>27 KB</b>	40×
LeNet-5 Ref	0.80%	-	1720 KB	
LeNet-5 Compressed	0.74%	-	<b>44 KB</b>	39×
AlexNet Ref	42.78%	19.73%	240 MB	
AlexNet Compressed	42.78%	19.70%	<b>6.9 MB</b>	35×
VGG-16 Ref	31.50%	11.32%	552 MB	
VGG-16 Compressed	31.17%	10.91%	<b>11.3 MB</b> 	49×

## 05. 실험결과

- Pruning과 Quantization을 함께 사용할 때 가장 성능이 우수
- 기법들을 개별적으로 사용할 때보다 압축 성능이 훨씬 좋음
- 복잡한 딥러닝 네트워크 모델을 모바일 앱에서 사용할 수 있게 해줌



Q & A