Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding

Han, Song, Huizi Mao, and William J. Dally

arXiv preprint arXiv:1510.00149 (2015).

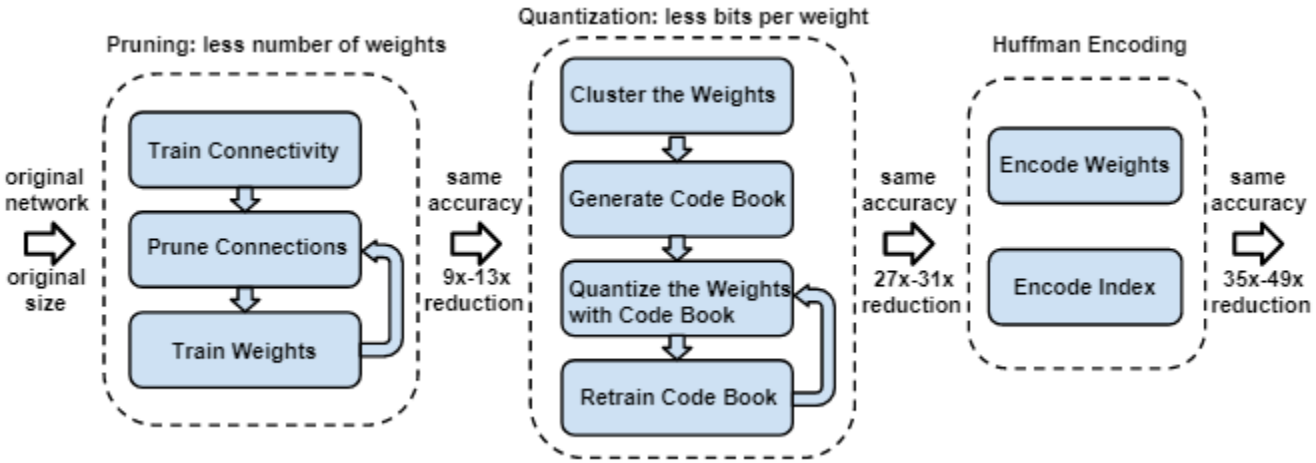
Introduction

- 모델이 더 높은 정확도를 얻기 위해 대형화 되면서
  모델의 가중치가 많은 저장공간과 메모리 대역폭을 소비하게 됨
  - Caffe 기준
    AlexNet: 약 200MB
    VGG16: 약 500MB

- 이러한 모델은 모바일 장치에서 활S용하기 어려움

Introduction

- 본 논문은 정확도를 유지하면서 신경망을 압축하기 위해 Pruning, Quantization, Huffman Coding을 사용



Network Pruning

- Han, Song, et al. "Learning both weights and connections for efficient neural network." *Advances in neural information processing systems*. 2015.

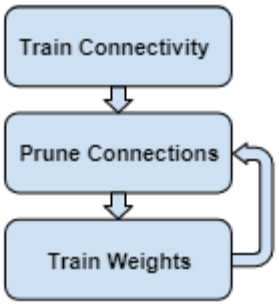


Figure 2: Three-Step Training Pipeline.

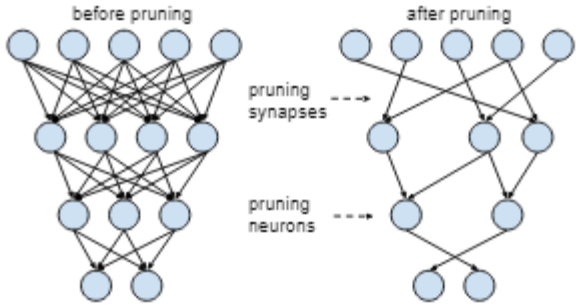


Figure 3: Synapses and neurons before and after pruning.

- 학습이 끝난 신경망에서 입계값 이하의 가중치를 가진 연결을 제거
- 남은 연결을 가지고 재 학습
- 이 과정을 반복하여 최소한의 연결을 가진 신경망을 확보

Network Pruning

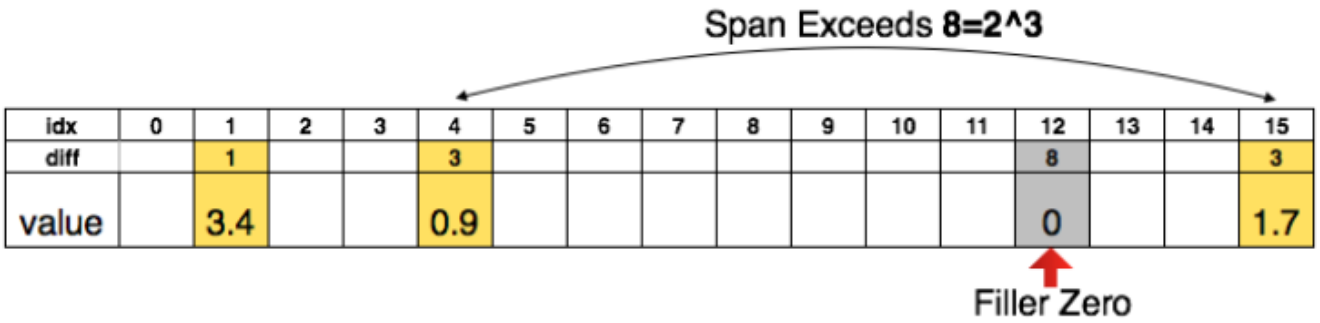
- Dropout Issue
  - Pruning을 적용한 신경망 모델은 이미 연결이 희소해졌기 때문에 기존의 Dropout 비율을 그대로 적용하면 안됨
  - 본 논문은 다음과 같이 Dropout 비율을 조정할 것을 제안

$$D_r = D_o \sqrt{\frac{C_{ir}}{C_{io}}}$$

  - D_r: 조정된 Dropout 비율
  - D_o: 기존 Dropout 비율
  - C_{ir}: Pruning 이후 노드간 연결의 수
  - C_{io}: 기존 노드간 연결의 수

Network Pruning

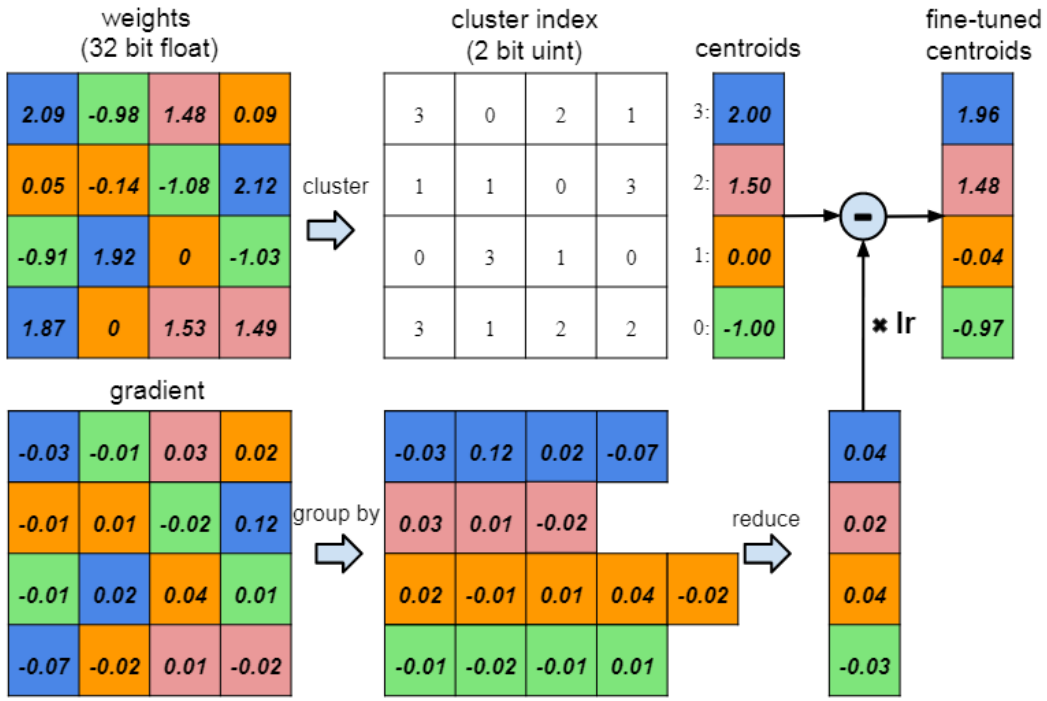
- Pruning 된 가중치를 더욱 압축하여 저장하기 위해
  인덱스 간의 거리를 저장



- 거리가 (예를 들어) 3비트를 초과할 경우,
  중간에 0 값을 추가해서 거리를 저장

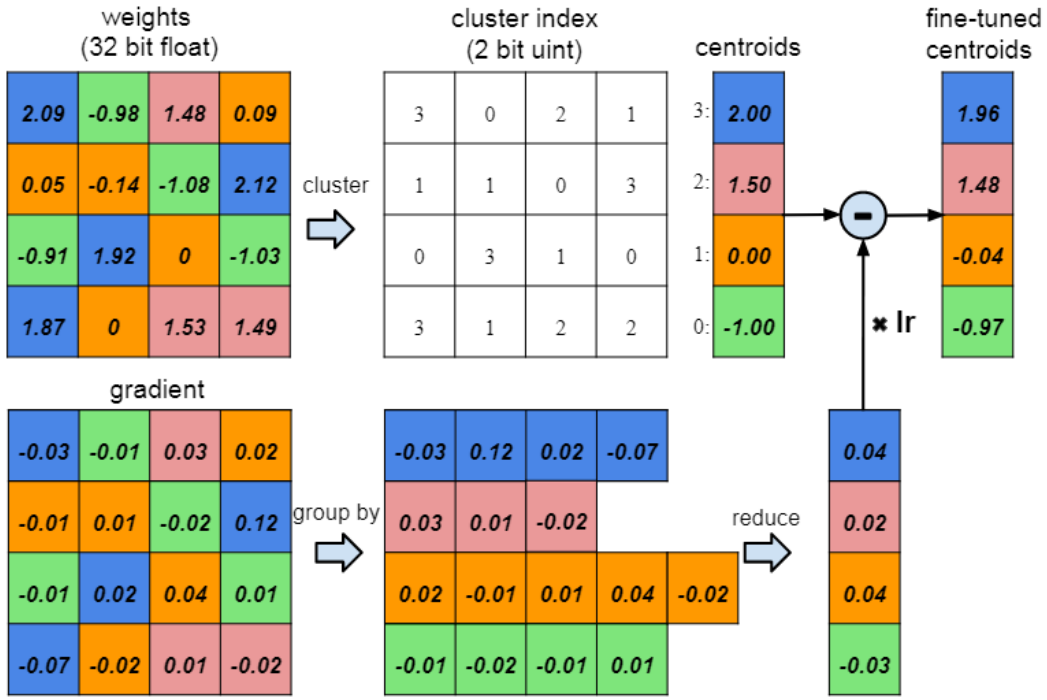
Trained Quantization and Weight Sharing

- K-means Clustering 이용해 Weight를 각 index별로 Quantization
- Gradient를 통해 fine-tuning



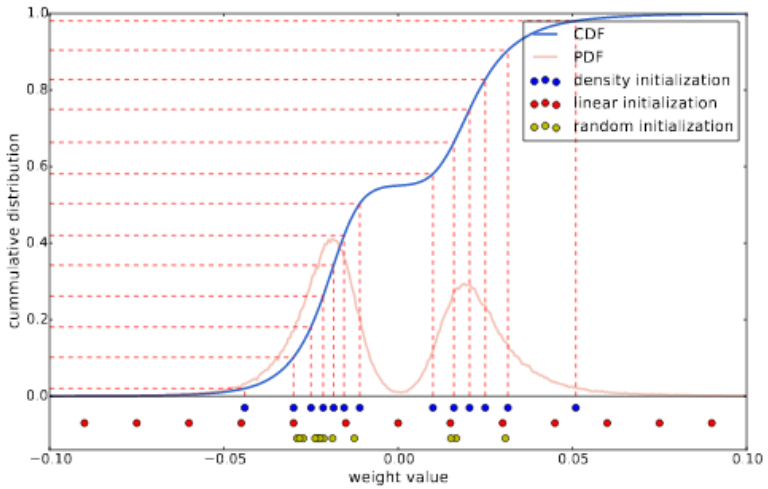
Trained Quantization and Weight Sharing

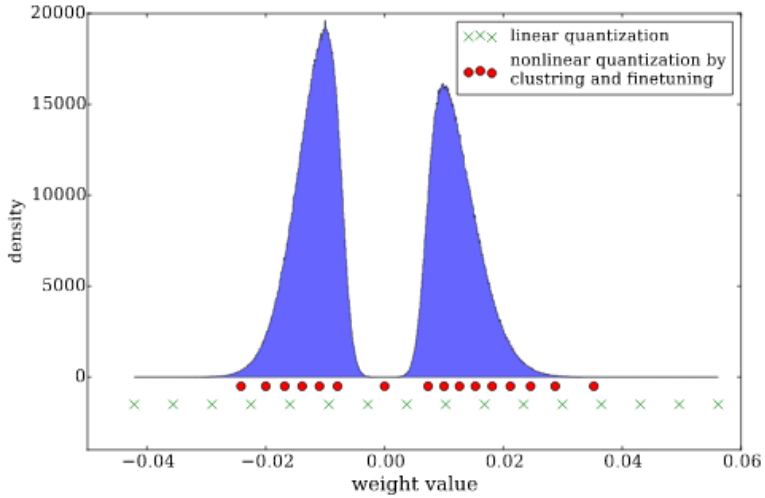
- Quantization 이전: $32\,bit \times 16 = 512bit$
- Quantization 이후: $32\,bit \times 4 + 2\,bit \times 16 = 160bit$
  - Quantization을 통해 3.2배 압축



Trained Quantization and Weight Sharing

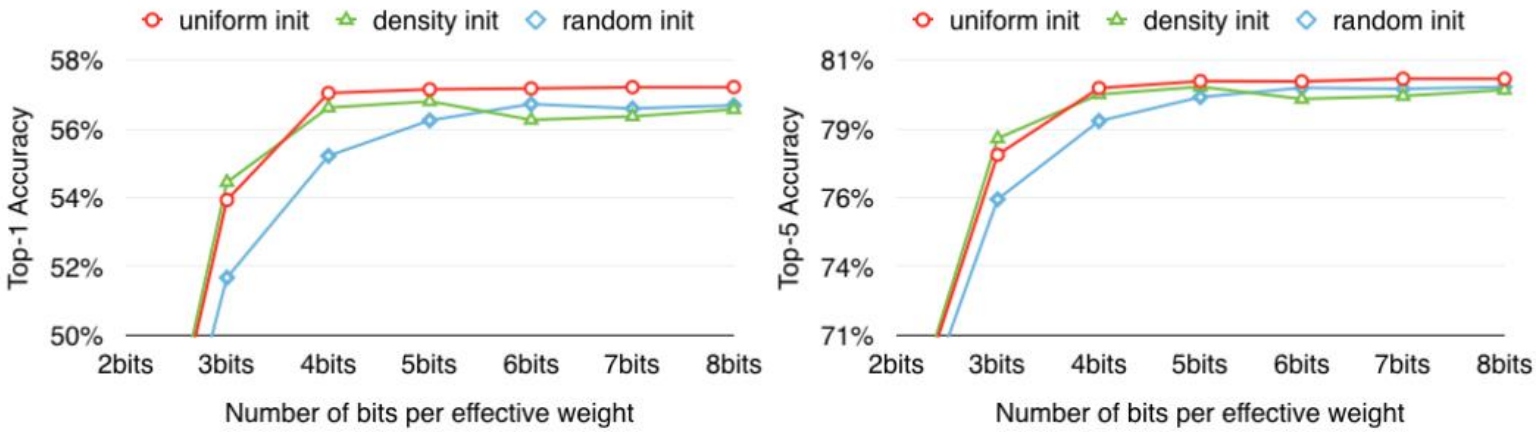
- Centroid Initialization을 위한 방법
  - Forgy(random), Density-based, Linear
  - Forgy와 Density-based 방식은 많이 나타나는 weight 값으로 집중되는 경향
    - 높은 가중치 값이 적게 나타나는 경우 무시될 수 있음
  - Linear 방식은 이런 문제를 피할 수 있음





Trained Quantization and Weight Sharing

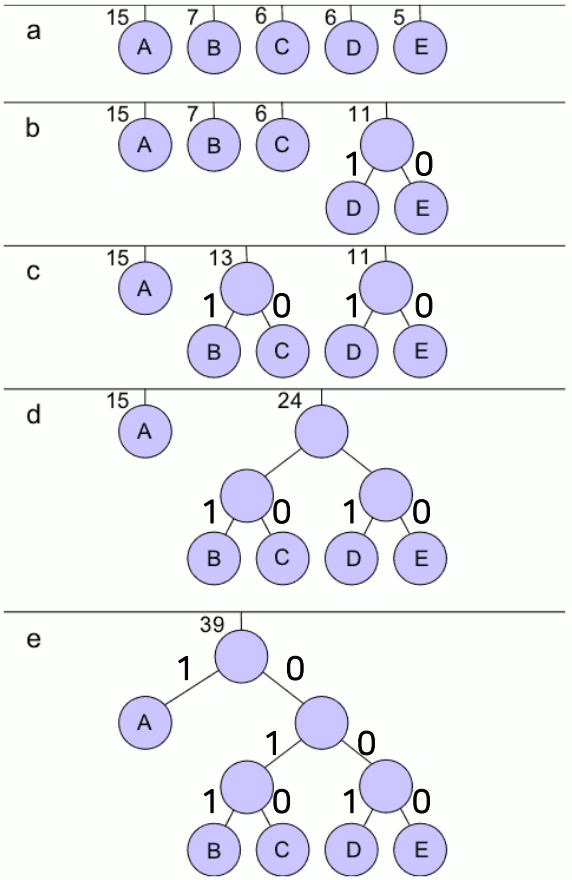
- 4bit Quantization까지는 Linear 방식이 더 높은 accuracy를 얻음
- 따라서 Linear 방식을 적용



Huffman Coding

- 자주 사용되는 값에 낮은 bit를 할당해 데이터를 압축하는 방식
  - A(15), B(7), C(6), D(6), E(5)
  - → A(1), B(011), C(010), D(001), E(000)

  - ACBCBCAAAAAEAAED $(8bit \times 16 = 128bit)$
  - → 1010011010011010111110001100000 1 $(32bit)$



Huffman Coding

- AlexNet의 마지막 FC 레이어를 보면
  Weight index와 Location index는 편향되어 있음

- Huffman coding을 통해 높은 압축 효율을 얻을 수 있음

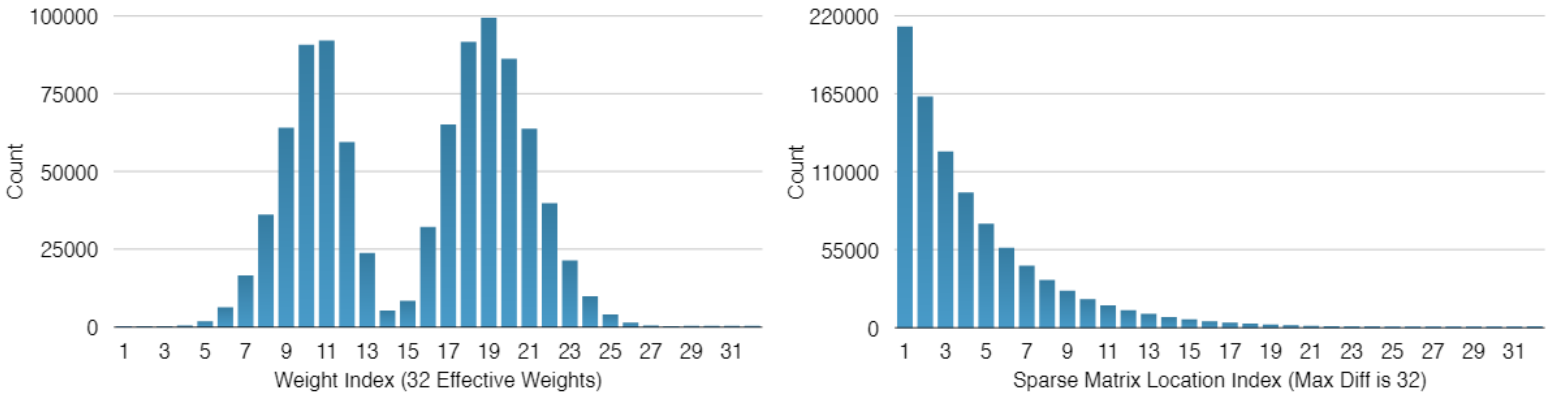


Figure 5: Distribution for weight (Left) and index (Right). The distribution is biased.

Experiments

- AlexNet

Table 4: Compression statistics for AlexNet. P: pruning, Q: quantization, H:Huffman coding.

Layer	#Weights	Weights% (P)	Weight bits (P+Q)	Weight bits (P+Q+H)	Index bits (P+Q)	Index bits (P+Q+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1	35K	84%	8	6.3	4	1.2	32.6%	20.53%
conv2	307K	38%	8	5.5	4	2.3	14.5%	9.43%
conv3	885K	35%	8	5.1	4	2.6	13.1%	8.44%
conv4	663K	37%	8	5.2	4	2.5	14.1%	9.11%
conv5	442K	37%	8	5.6	4	2.5	14.0%	9.43%
fc6	38M	9%	5	3.9	4	3.2	3.0%	2.39%
fc7	17M	9%	5	3.6	4	3.7	3.0%	2.46%
fc8	4M	25%	5	4	4	3.2	7.3%	5.85%
Total	61M	11%(9×)	5.4	4	4	3.2	3.7% (**27**×)	2.88% (**35**×)

Experiments

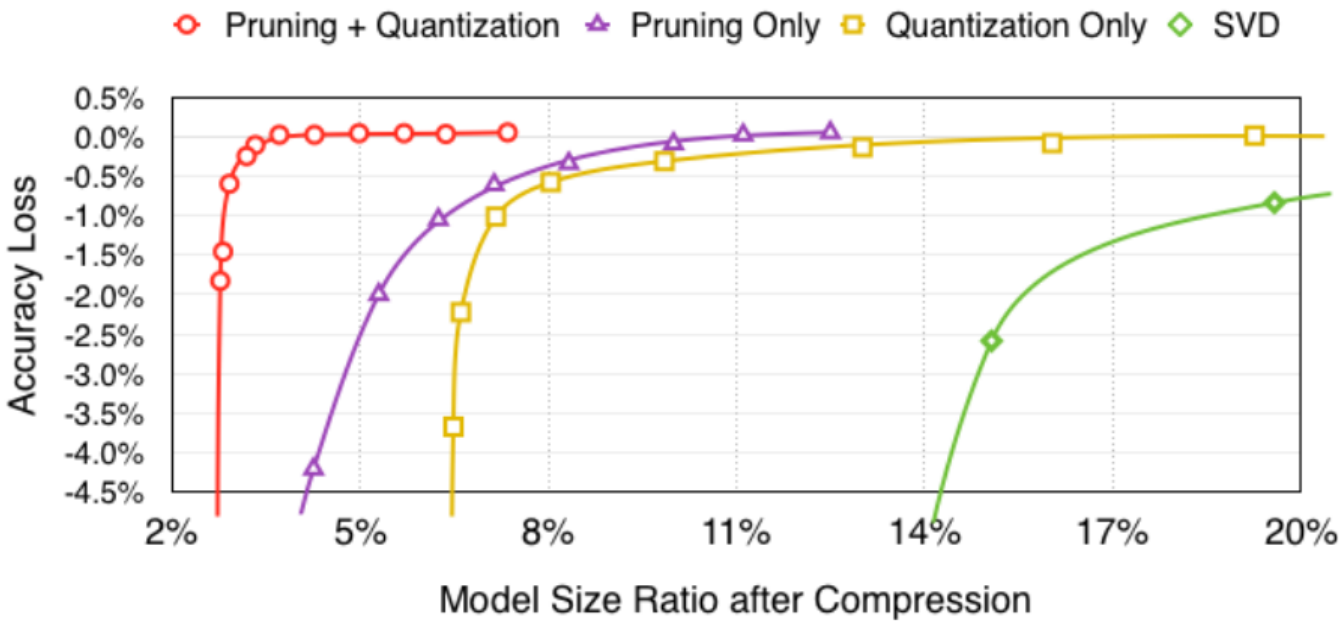
- VGG16

Table 5: Compression statistics for VGG-16. P: pruning, Q:quantization, H:Huffman coding.

Layer	#Weights	Weights% (P)	Weigh bits (P+Q)	Weight bits (P+Q+H)	Index bits (P+Q)	Index bits (P+Q+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1_1	2K	58%	8	6.8	5	1.7	40.0%	29.97%
conv1_2	37K	22%	8	6.5	5	2.6	9.8%	6.99%
conv2_1	74K	34%	8	5.6	5	2.4	14.3%	8.91%
conv2_2	148K	36%	8	5.9	5	2.3	14.7%	9.31%
conv3_1	295K	53%	8	4.8	5	1.8	21.7%	11.15%
conv3_2	590K	24%	8	4.6	5	2.9	9.7%	5.67%
conv3_3	590K	42%	8	4.6	5	2.2	17.0%	8.96%
conv4_1	1M	32%	8	4.6	5	2.6	13.1%	7.29%
conv4_2	2M	27%	8	4.2	5	2.9	10.9%	5.93%
conv4_3	2M	34%	8	4.4	5	2.5	14.0%	7.47%
conv5_1	2M	35%	8	4.7	5	2.5	14.3%	8.00%
conv5_2	2M	29%	8	4.6	5	2.7	11.7%	6.52%
conv5_3	2M	36%	8	4.6	5	2.3	14.8%	7.79%
fc6	103M	4%	5	3.6	5	3.5	1.6%	1.10%
fc7	17M	4%	5	4	5	4.3	1.5%	1.25%
fc8	4M	23%	5	4	5	3.4	7.1%	5.24%
Total	138M	7.5%(13×)	6.4	4.1	5	3.1	3.2% (**31**×)	2.05% (**49**×)

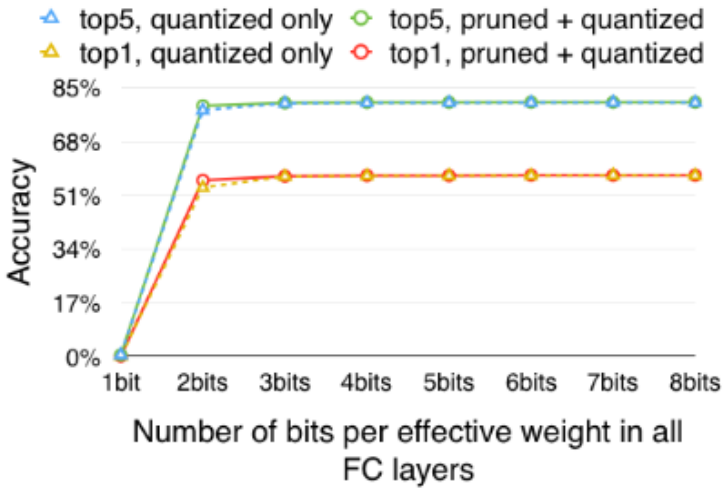
Discussions

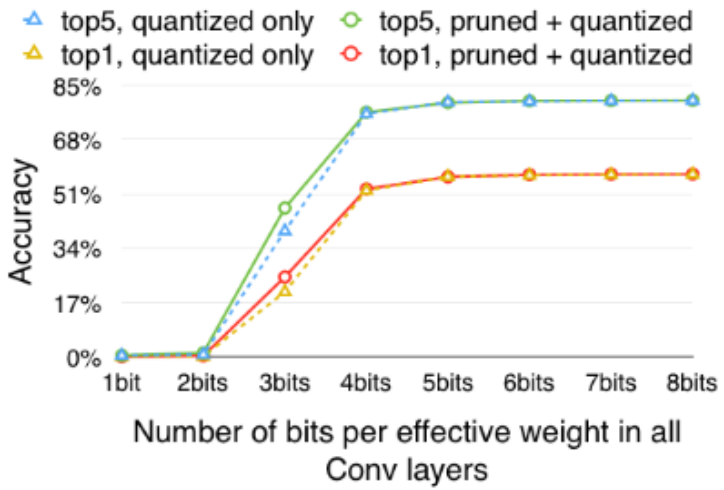
- Pruning, Quantization 효율 비교



Discussions

- Pruning, Quantization 효율 비교





  - FC 레이어가 더 robust함

Discussions

- Quantization에 따른 error rate 비교 (AlexNet)

#CONV bits / #FC bits	Top-1 Error	Top-5 Error	Top-1 Error Increase	Top-5 Error Increase
32bits / 32bits	42.78%	19.73%	-	-
8 bits / 5 bits	42.78%	19.70%	0.00%	-0.03%
8 bits / 4 bits	42.79%	19.73%	0.01%	0.00%
4 bits / 2 bits	44.77%	22.33%	1.99%	2.60%

  - 8bit / 4bit Quantization까지는 성능 저하가 거의 없음
  - 약간의 성능 저하를 허용한다면 더 공격적인 모델 압축을 할 수 있음

Discussions

- 본 논문의 방법이 다른 압축 방법보다 높은 효율을 얻음

Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
Baseline Caffemodel (BVLC)	42.78%	19.73%	240MB	1×
Fastfood-32-AD (Yang et al., 2014)	41.93%	-	131MB	2×
Fastfood-16-AD (Yang et al., 2014)	42.90%	-	64MB	3.7×
Collins & Kohli (Collins & Kohli, 2014)	44.40%	-	61MB	4×
SVD (Denton et al., 2014)	44.02%	20.56%	47.6MB	5×
Pruning (Han et al., 2015)	42.77%	19.67%	27MB	9×
Pruning+Quantization	42.78%	19.70%	8.9MB	27×
Pruning+Quantization+Huffman	**42.78%**	**19.70%**	**6.9MB**	**35×**

Conclusion

- 본 논문은 Pruning, Quantization, Huffman coding을 통해
  정확도 손실 없이 모델 가중치를 AlexNet은 35배, VGG16은 49배 줄임

- 이러한 방법을 통해 메모리 공간이 제한된 환경에서도
  복잡한 신경망을 사용할 수 있음