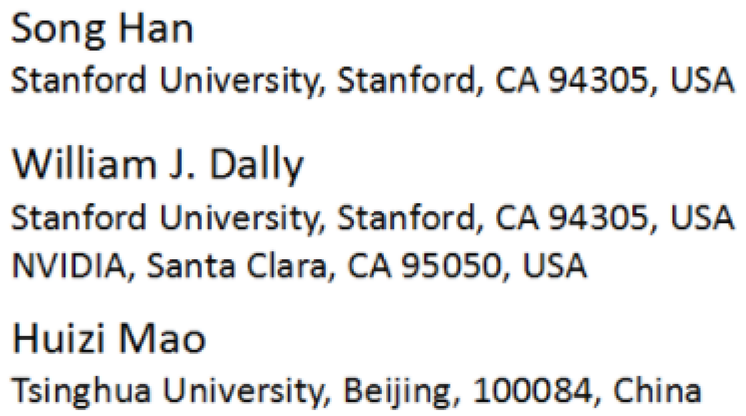
DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN


Song Han
Stanford University, Stanford, CA 94305, USA

William J. Dally
Stanford University, Stanford, CA 94305, USA
NVIDIA, Santa Clara, CA 95050, USA

Huizi Mao
Tsinghua University, Beijing, 100084, China

ICLR2016 best paper

Freya

20201122

Contents

Abstract

Why?

Neural networks are difficult to deploy on embedded systems with limited hardware resources.

- Computationally intensive
- Memory intensive

How?

Prunes the network：learning only the important connections, 9× to 13× reduction

Quantize and share the weights:quantize the weights to enforce weight sharing, 32 bit to 5 bit

Huffman coding: saves 20% − 30% of network storage.

Result?

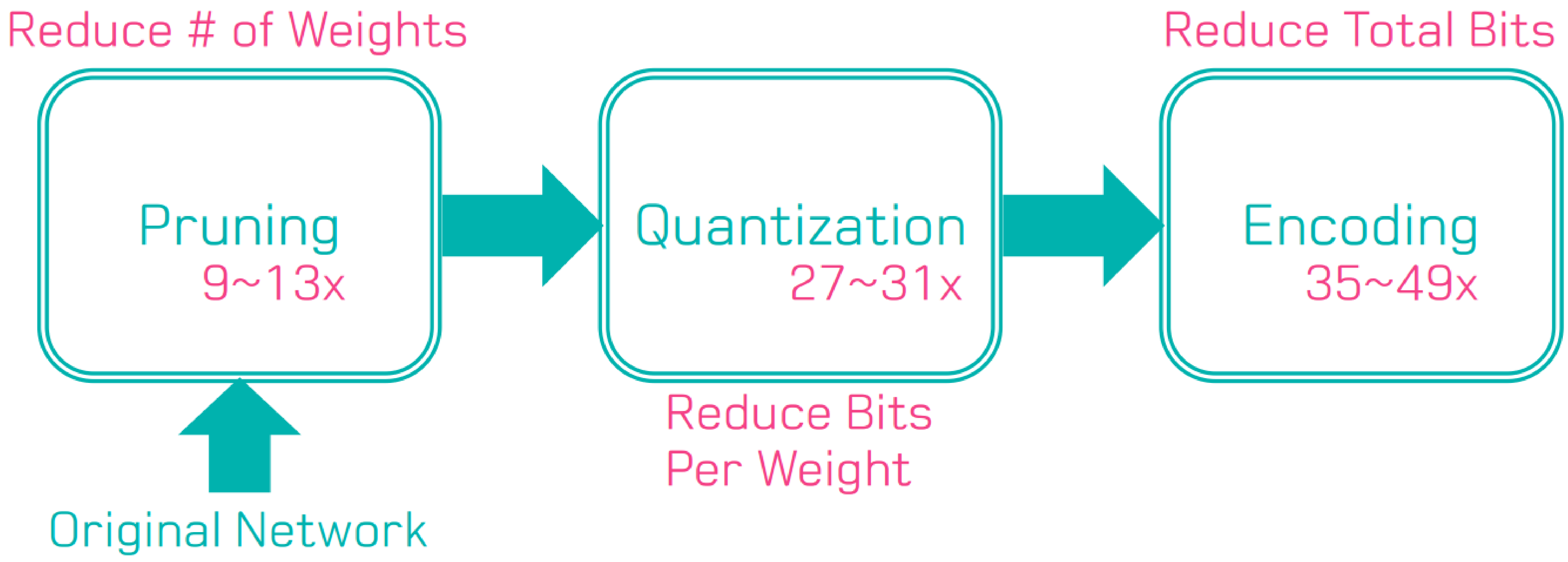
AlexNet by 35× , from 240MB to 6.9MB, without loss of accuracy

VGG-16 by 49× from 552MB to 11.3MB, without loss of accuracy

3× to 4× layerwise speedup

3× to 7× better energy efficiency

Framework


Reduce # of Weights
Reduce Total Bits
Pruning
9~13x
Quantization
27~31x
Encoding
35~49x
Reduce Bits
Per Weight
Original Network

Methods

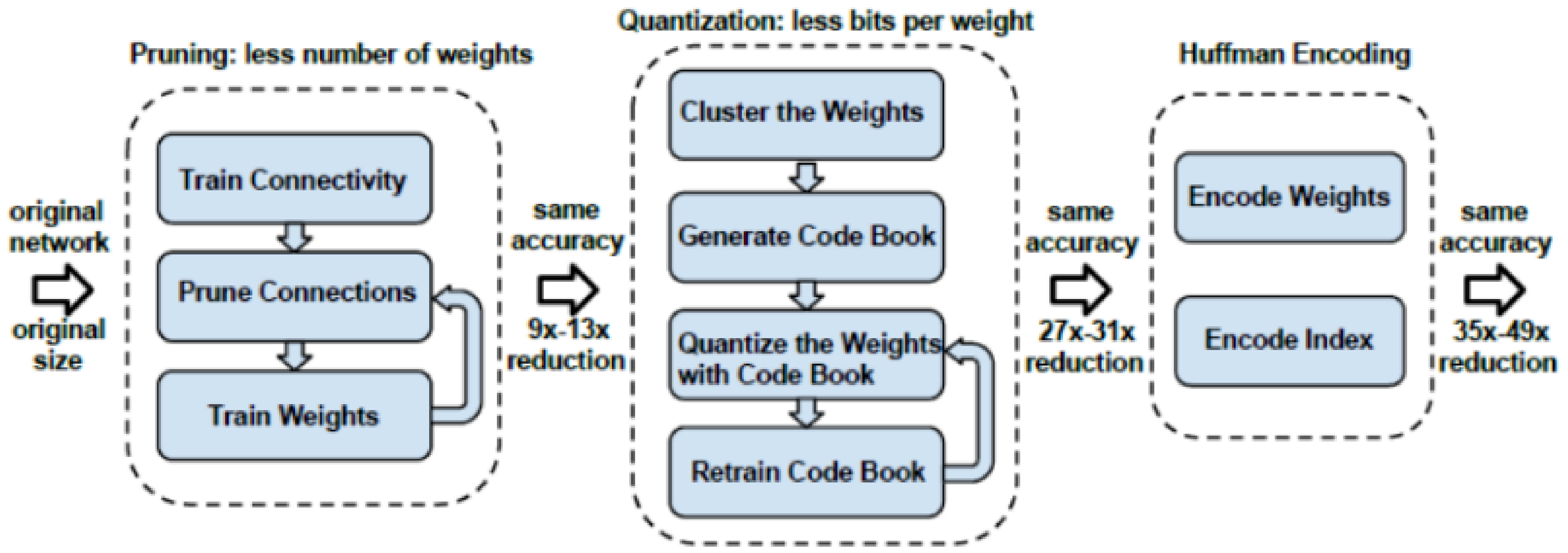
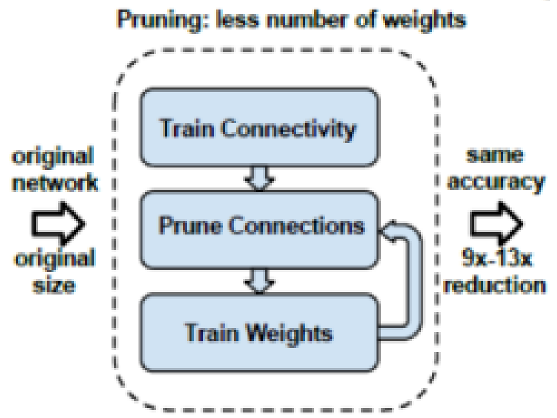


Figure 1: The three stage compression pipeline: pruning, quantization and Huffman coding. Pruning reduces the number of weights by 10×, while quantization further improves the compression rate: between 27× and 31×. Huffman coding gives more compression: between 35× and 49×. The compression rate already included the meta-data for sparse representation. The compression scheme doesn't incur any accuracy loss.

Pruning



1. Learning the connectivity via normal network training.

2. We prune the small-weight connections: all connections with weights below a threshold are removed from the network.

3. We retrain the network to learn the final weights for the remaining sparse connections.

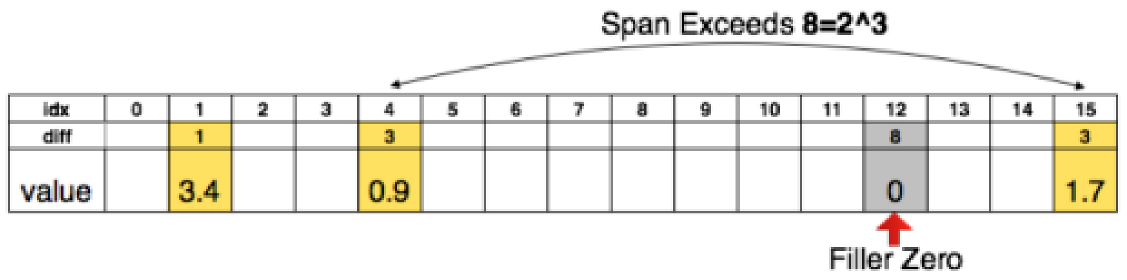
Using compressed sparse row (CSR) or compressed sparse column (CSC) format

$$A = \begin{array}{ccccc} 4.0 & 1.0 & 0.0 & 0.0 & 2.5 \\ 0.0 & 4.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 4.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 & 4.0 & 0.0 \\ 2.5 & 0.0 & 0.0 & 0.5 & 4.0 \end{array}$$

2a+n+1 numbers

```
AA  =  4.0  1.0  2.5  4.0  1.0  1.0  4.0  1.0  1.0  4.0  2.5  0.5  4.0
JA  =  1    4    6    9    11   14
IC  =  1    2    5    2    3    2    3    5    3    4    1    4    5
```

Store the index difference instead of the absolute position



idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
diff		1			3								8			3
value		3.4			0.9								0			1.7

Figure 2: Representing the matrix sparsity with relative index. Padding filler zero to prevent overflow.

Weight Shared & Quantization

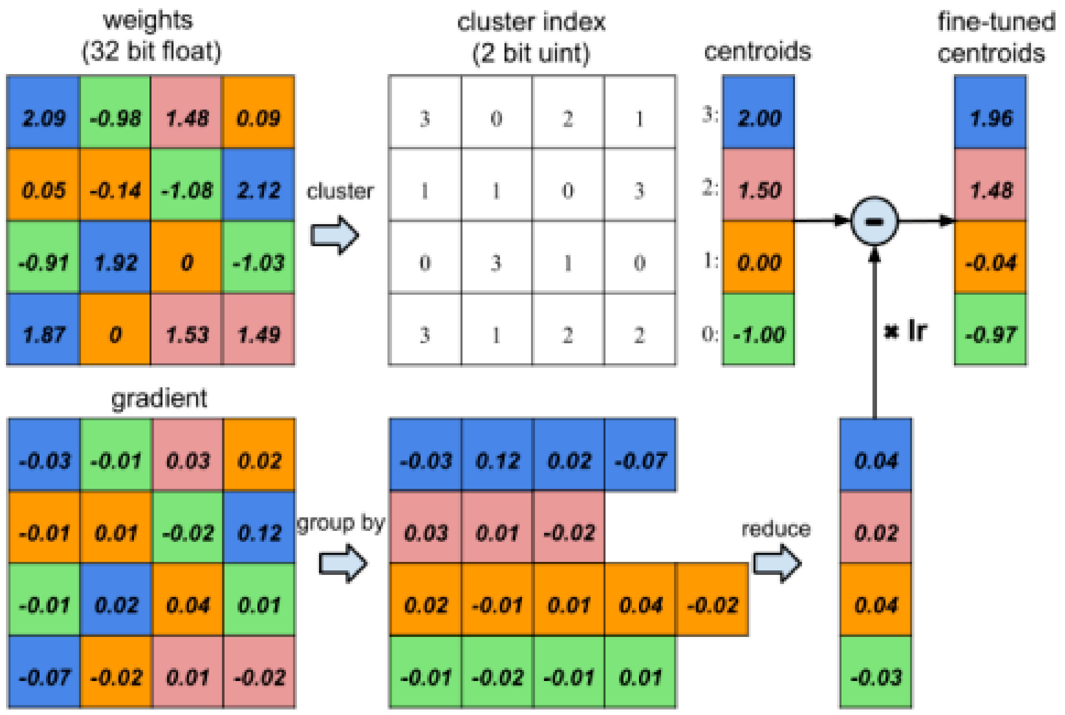


Figure 3: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom).

Weight sharing

k-means clustering for each layer

within-cluster sum of squares (WCSS) :

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

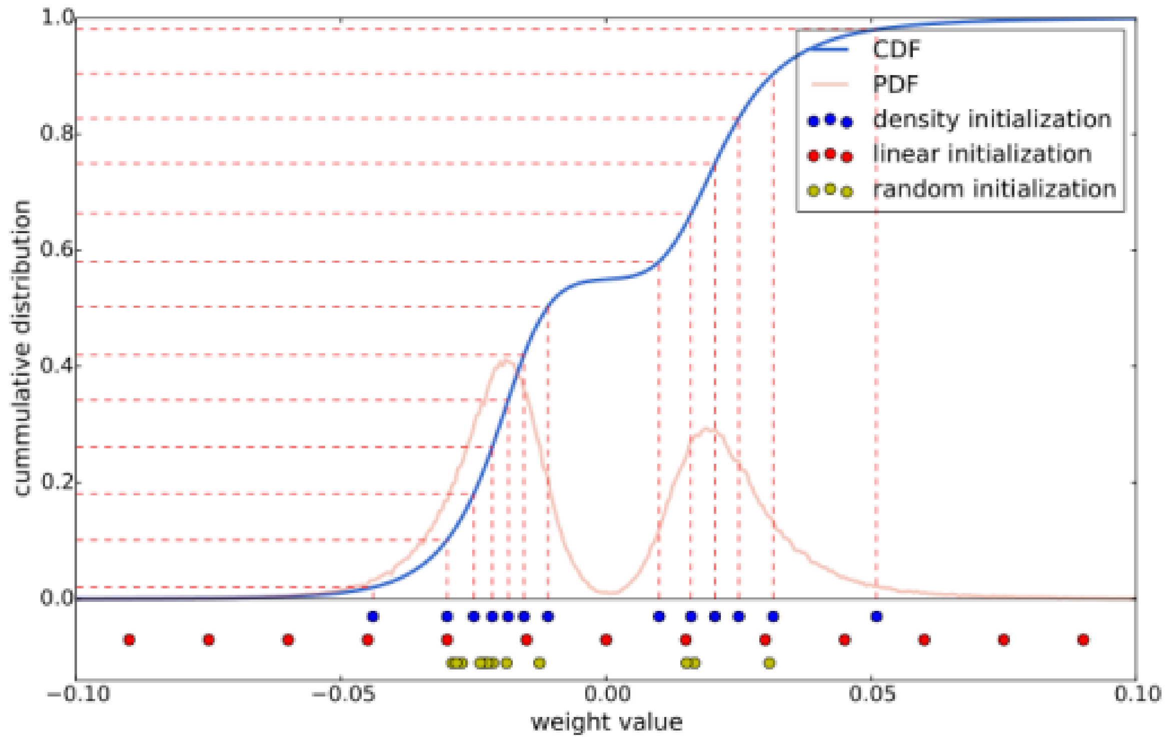
1.Reducing the number of bits required to represent each weight.

2.Limit the number of effective weights we need to store by having multiple connections share the same weight.

3.Fine-tune those shared weights.

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

Initialization of shared weights



Random: Initialization randomly chooses k observations from the data set and uses these as the initial centroids.

Density-based initialization: Linearly spaces the CDF of the weights in the y-axis, then finds the horizontal intersection with the CDF, and finally finds the vertical intersection on the x-axis.

Linear initialization: Linearly spaces the centroids between the [min, max] of the original weights.

Larger weights play a more important role than smaller weights,The experiment section compares the accuracy showing that linear initialization works best.

Huffman coding

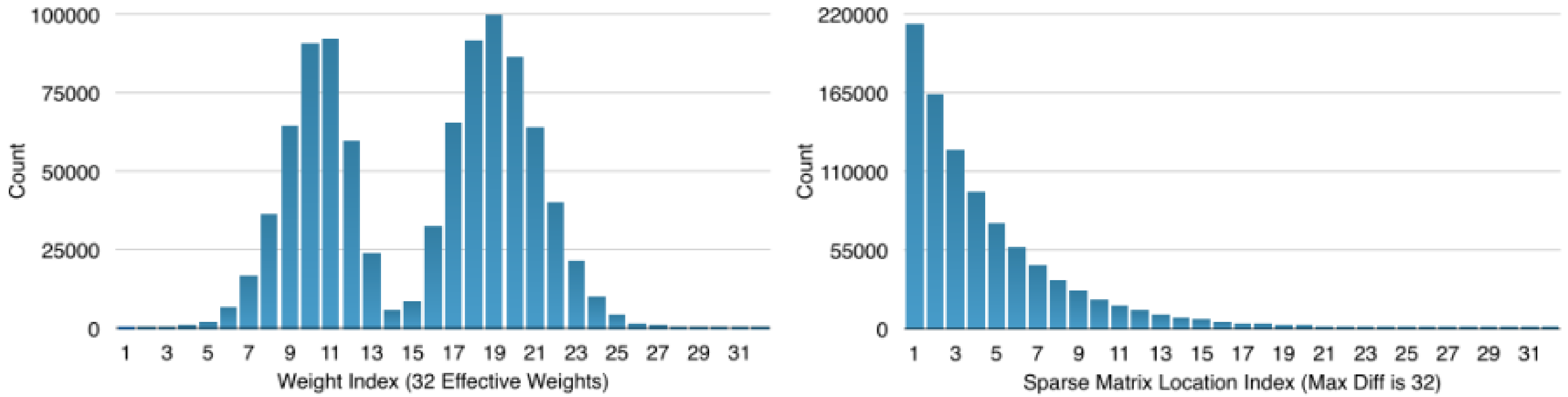
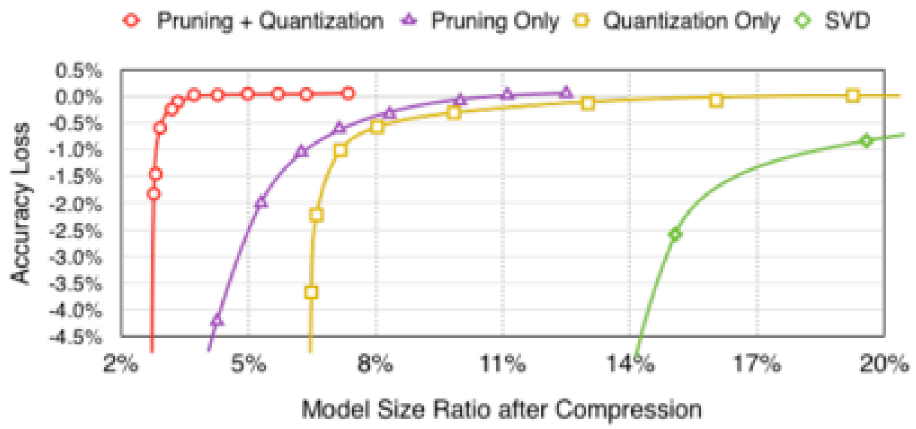
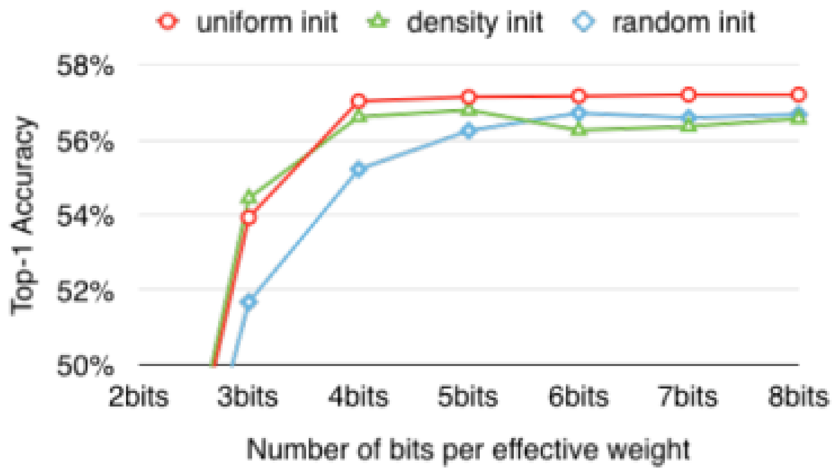


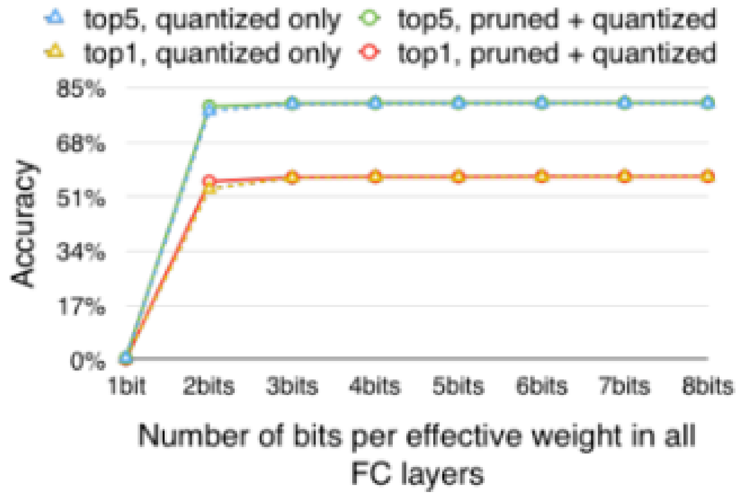
Figure 5: Distribution for weight (Left) and index (Right). The distribution is biased.

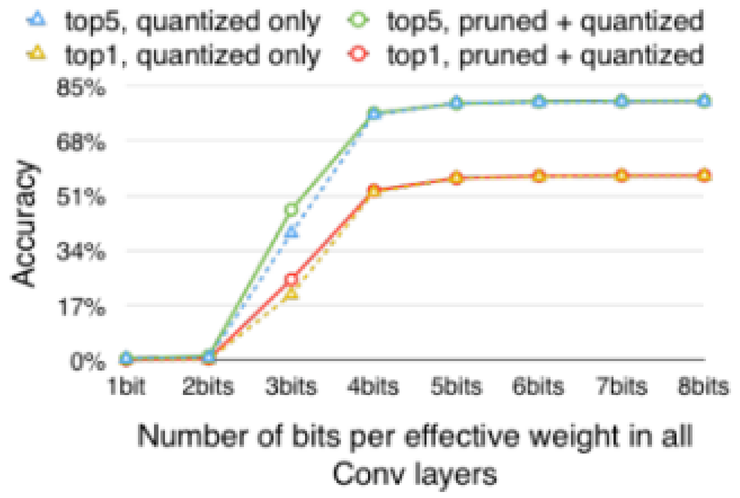
Figure 5 : shows the probability distribution of quantized weights and the sparse matrix index of the last fully connected layer in AlexNet. Both distributions are biased: most of the quantized weights are distributed around the two peaks; the sparse matrix index difference are rarely above 20. Experiments show that Huffman coding these non-uniformly distributed values saves 20% – 30% of network storage.

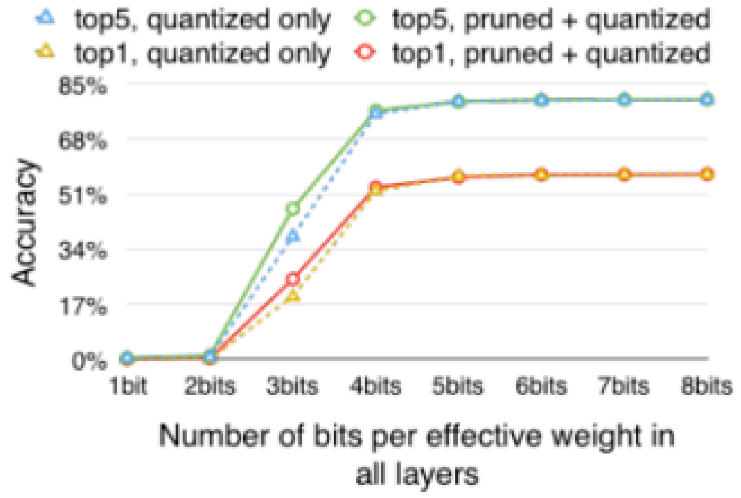
Experiments



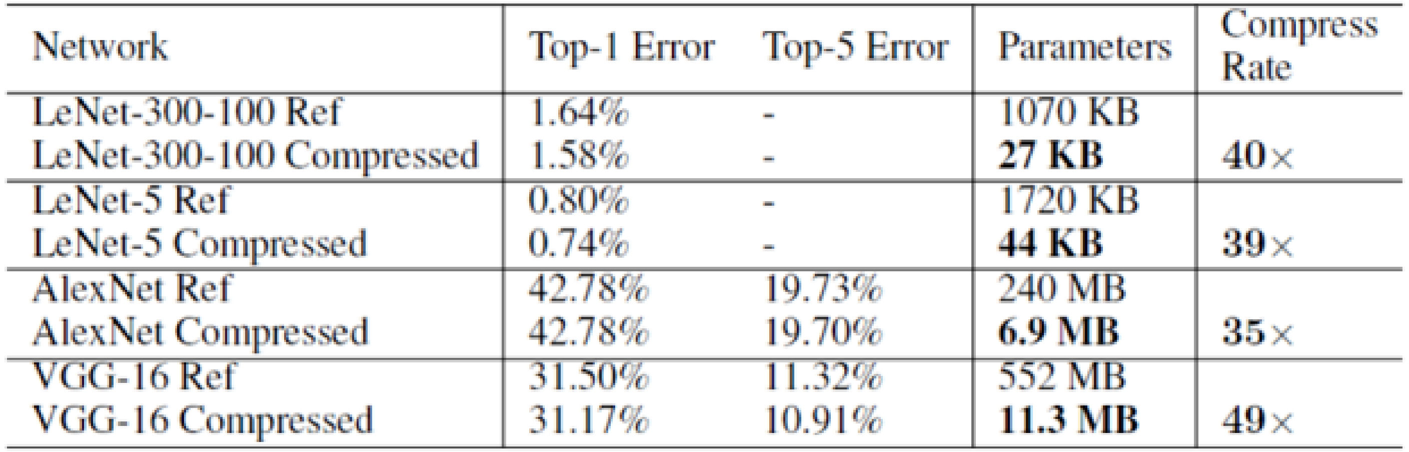




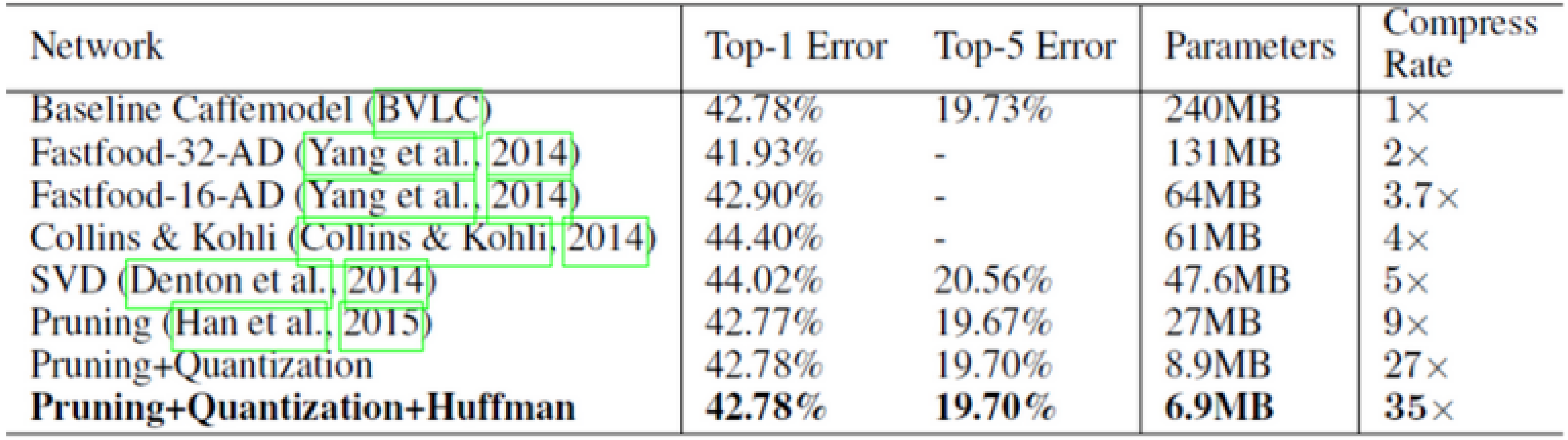




Experiments



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



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

01 Simple methods can achieve very good performance.

02 This potentially makes deep neural networks more energy efficient to run on mobile.

03 Many model and tasks can be applied on mobile and embedded devices

History

- Song Han et al., Learning both Weights and Connections for Efficient Neural Networks, *NIPS 2015*
- Song Han et al., Deep Compression, *ICLR 2016 Best Paper*
- Song Han et al,. EIE: Efficient Inference Engine on Compressed Deep Neural Network, *ISCA 2016*
- Song Han et al., DSD: Dense-Sparse-Dense Training for Deep Neural Networks, *ICLR 2017*

https://cs.nyu.edu/~fergus/teaching/vision/9a_song.pdf