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

Hu Tao

Peking University

Acknowledge: some of figures are based on [Han song talk at gputechconf]

Author Info

- Song Han, Huizi Mao, William J. Dally
- Stanford University
- ICLR 2016 Best Paper Reward

Overview

- Three stages
- Experiment
- Background
- Thinking

Three stages

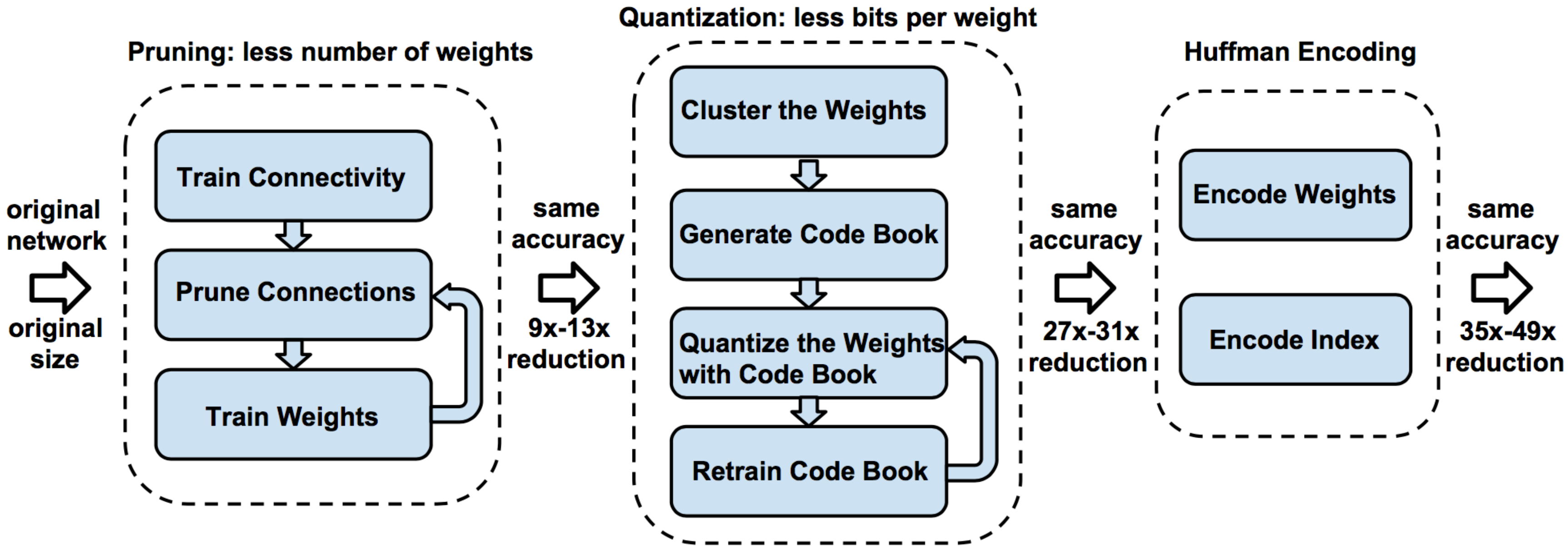
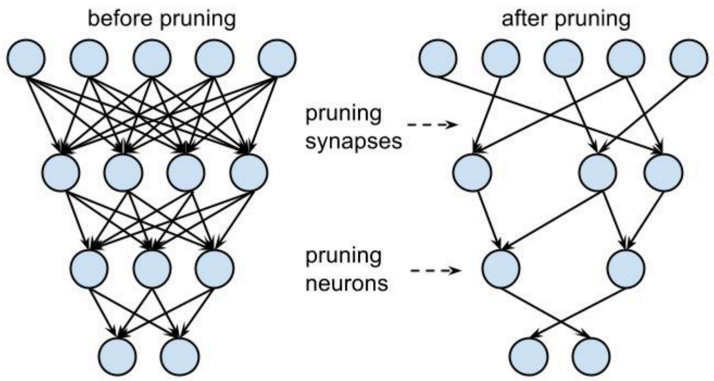


Figure 1: The three stage compression pipeline: pruning, quantization and Huffman coding. Pruning reduces the number of weights by $10\times$, while quantization further improves the compression rate: between $27\times$ and $31\times$. Huffman coding gives more compression: between $35\times$ and $49\times$.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

Stage one: pruning

---Less number of weights

- How-to-compress



- How-to-compensate

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

Stage one:　pruning

---Less number of weights

- How to store sparse weight matrix

- compressed sparse row (CSR)

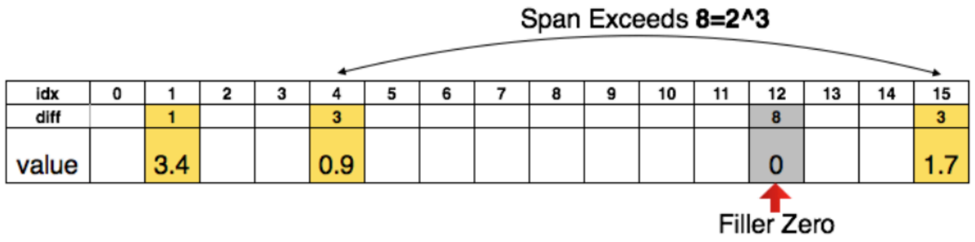


Figure 2: Representing the matrix sparsity with relative index. Padding filler zero to prevent overflow.

Stage one: pruning

---Less number of weights

- Fc layer is ok, what about convolution layer?

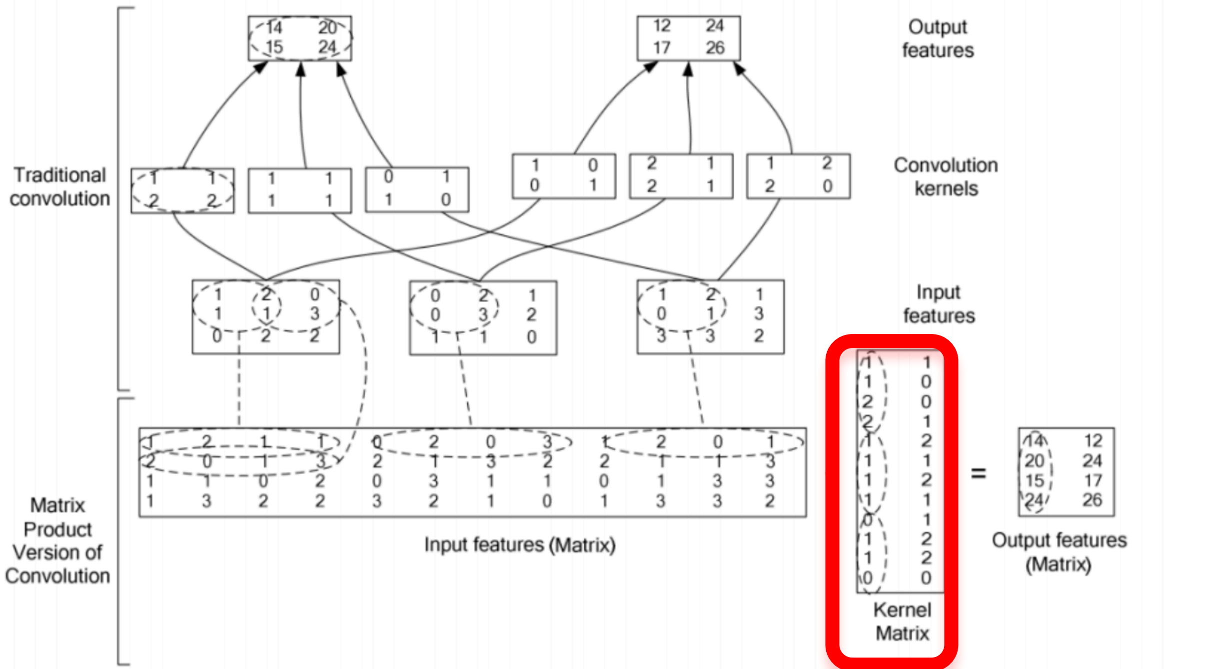


Figure 2. Example convolution operations in a convolutional layer (biases, sub-sampling, and non-linearity omitted). The top figure presents the traditional convolution operations, while the bottom figure presents the matrix version.

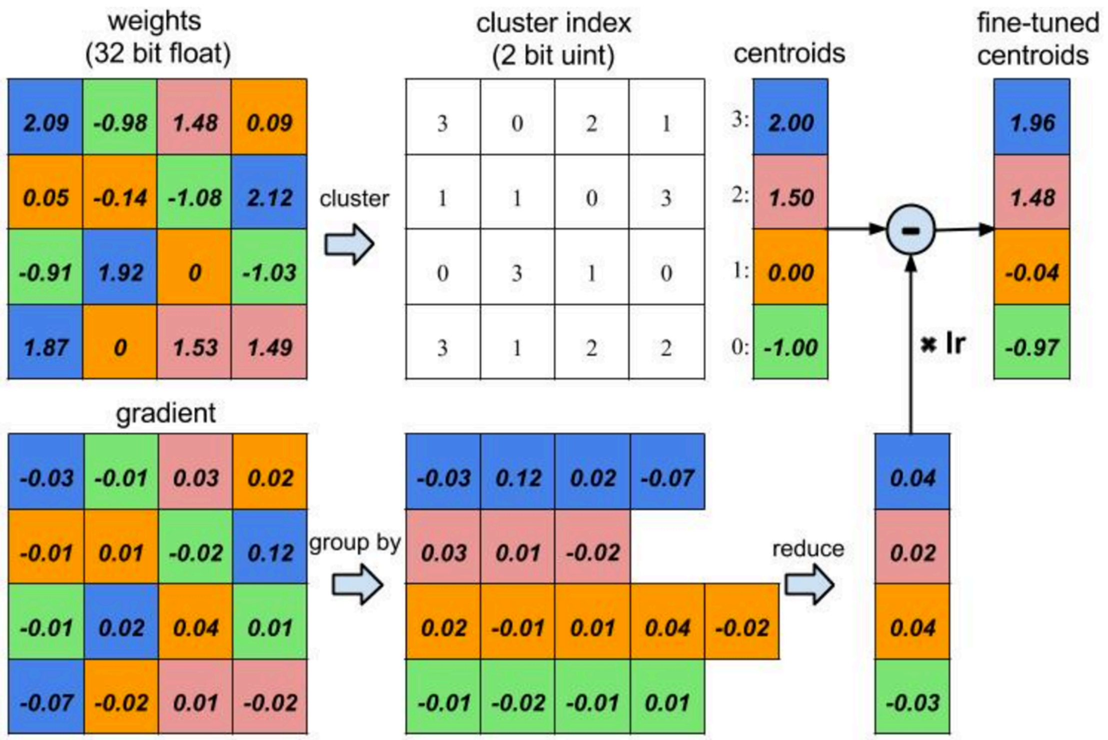
GEMM(GEneral Matrix to Matrix Multiplication) is at the heart of deep learning!

Stage two: quantization

---Less bits per weights

- How-to-compress

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



- How-to-compensate

fine-tune those shared weights.

Stage two: quantization

---Less bits per weights

- Initialization of K-means

1,Density init

2,Linear init

3,Random init

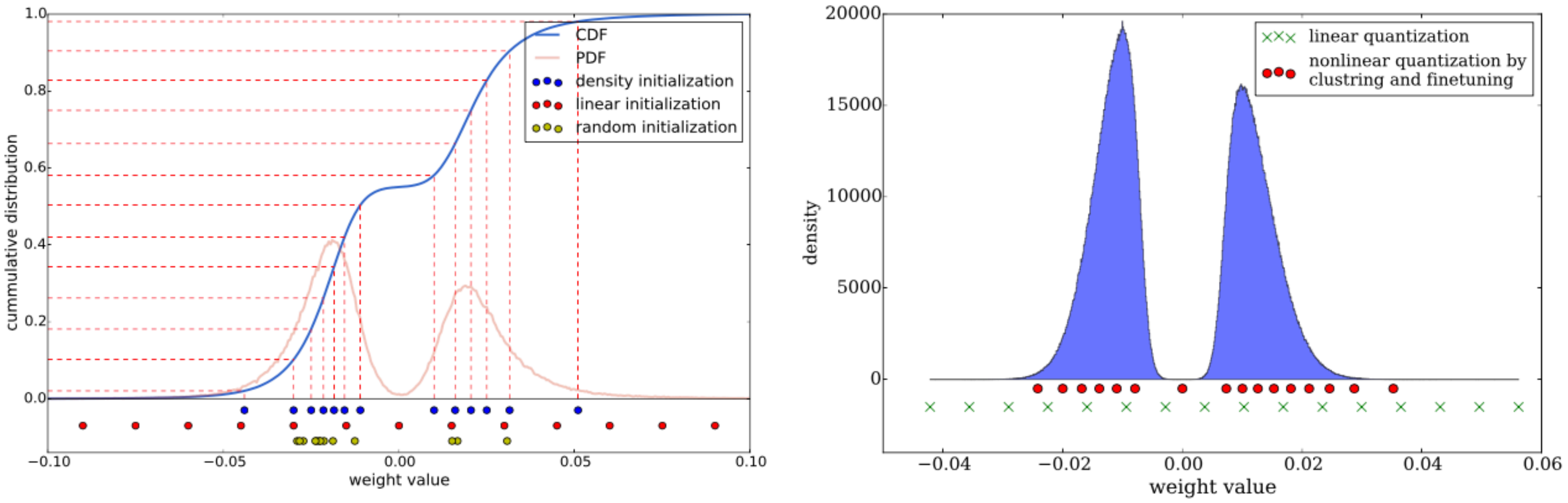


Figure 4: Left: Three different methods for centroids initialization. Right: Distribution of weights (blue) and distribution of codebook before (green cross) and after fine-tuning (red dot).

Stage two: quantization

---Less bits per weights

- Back propagation

We denote the loss by $\mathcal{L}$, the weight in the ith column and jth row by W_{ij}, the centroid index of element $W_{i,j}$ by I_{ij}, the kth centroid of the layer by C_k. By using the indicator function $\mathbb{1}(.)$, the gradient of the centroids is calculated as:

$$\frac{\partial \mathcal{L}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \frac{\partial W_{ij}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \mathbb{1}(I_{ij} = k) \tag{3}$$

Stage Three: Huffman Coding

- The table is derived from the occurrence probability for each symbol. More common symbols are represented with fewer bits.

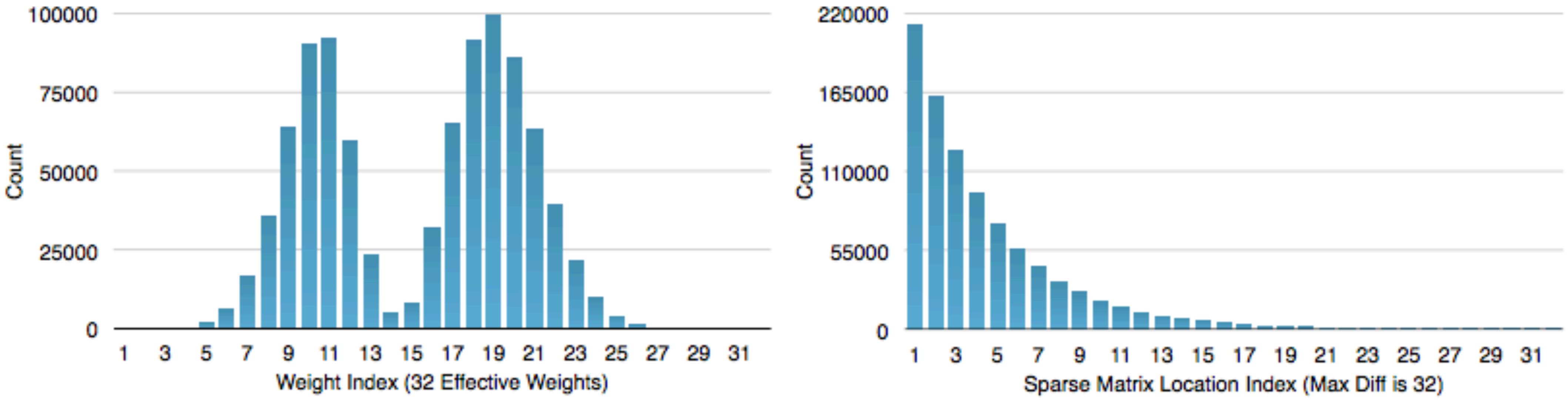


Figure 5: Distribution for weight (Left) and index (Right). The distribution is biased.

of different initialization methods after clustering and fine-tuning, showing that linear initialization works best.

Three Stages Summary

- Pruning: CSR Format to store sparse matrix
- Quantization: Codebook, Weight, Index
- Huffman

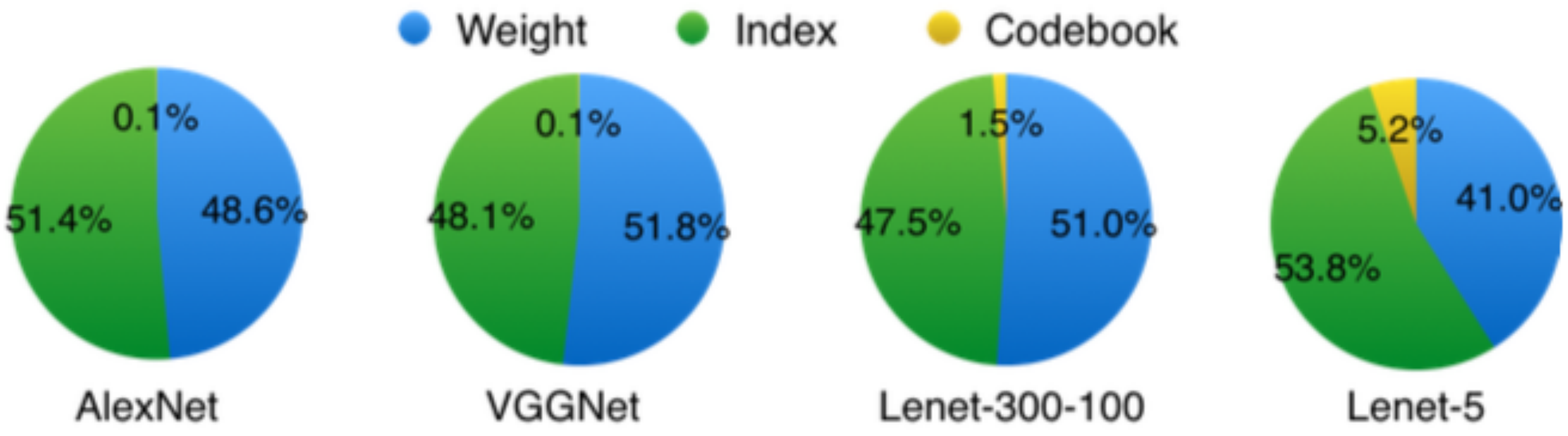


Figure 11: Storage ratio of weight, index and codebook.

How to run in chips?

- Huffman Code translation
- Look up codebook to acquire the actual weight
- Calculate actual position according the relative index
- Prepare the multiplication matrix
- Forward pass

Experiment

Table 1: The compression pipeline can save 35× to 49× parameter storage with no loss of accuracy.

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×**

Experiment

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**×)

Experiment—Speed and Energy

Sparse-on-GPU: cuSPARSE CSRMV

Dense-on-GPU: cuBLAS GEMV

Sparse-on-CPU: MKL SPBLAS CSRMV

Sparse-on-CPU: MKL CBLAS GEMV

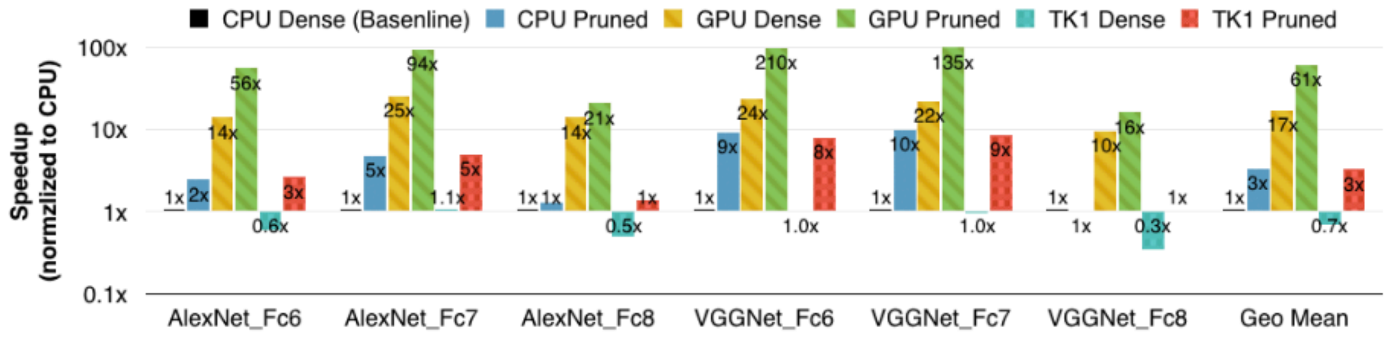


Figure 9: Compared with the original network, pruned network layer achieved 3× speedup on CPU, 3.5× on GPU and 4.2× on mobile GPU on average. Batch size = 1 targeting real time processing. Performance number normalized to CPU.

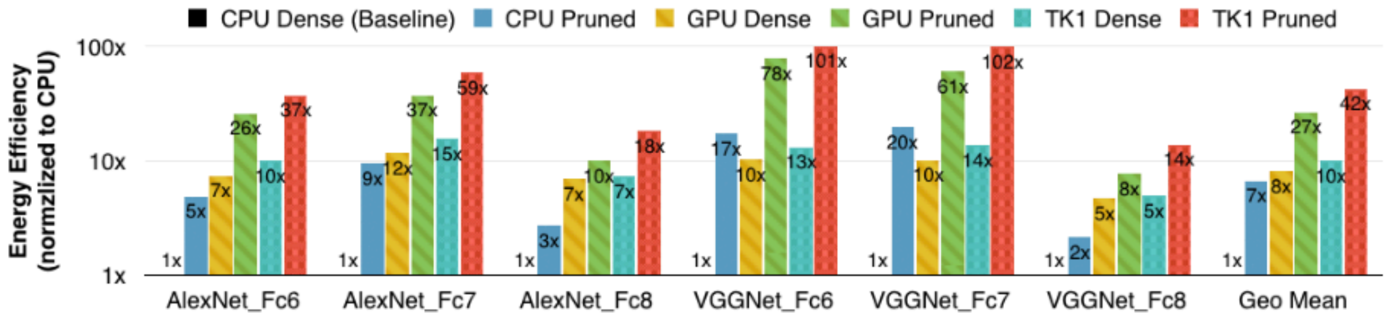


Figure 10: Compared with the original network, pruned network layer takes 7× less energy on CPU, 3.3× less on GPU and 4.2× less on mobile GPU on average. Batch size = 1 targeting real time processing. Energy number normalized to CPU.

Back to 1 Oct 2015

- **FIXED POINT OPTIMIZATION OF DEEP CONVOLUTIONAL NEURAL NETWORKS FOR OBJECT RECOGNITION(ICASSP2015)** Quantization layer by layer, achieve compression and accuracy increase
- **Fixed-Point Feedforward Deep Neural Network Design Using Weights +1, 0, and -1 (SiPS2014)**
- Quantization with weight 1,0,-1,a good design of back-propagation

Reference

[1]. https://www.oreilly.com/ideas/compressing-and-regularizing-deep-neural-networks

[2].https://www.youtube.com/watch?time_continue=24&v=vouEMwDNopQ

[3].https://en.wikipedia.org/wiki/Sparse_matrix#Compressed_sparse_row_.28CSR.2C_CRS_or_Yale_format.29

Ideas

- Fix Point Continuation,扩展DSD
- (更好的极值点，更高的压缩比)