DEEP COMPRESSION:

COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING

Authors: Song Han
Huizi Mao
William J. Dally

Presenters: Lingjiao Chen
Jinman Zhao
Charles Kuang

Neural Network is so important and widely-used...

Go player

Image classification

Natural language translation

Program Generation

Automatic driving

……

But...

1. Neural Network is typically very large

   AlexNet: 200 MB+

   VGG-16: 500 MB+

2. It also consumes a lot of energy.

How can we make neural network suitable for mobile applications?

Deep Compression!

Idea: Compress the neural network without losing accuracy!

Reference: Han et. al. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, ICLR 2016

Note: This paper received the **best paper award** in ICLR 2016.

Key Idea Overview

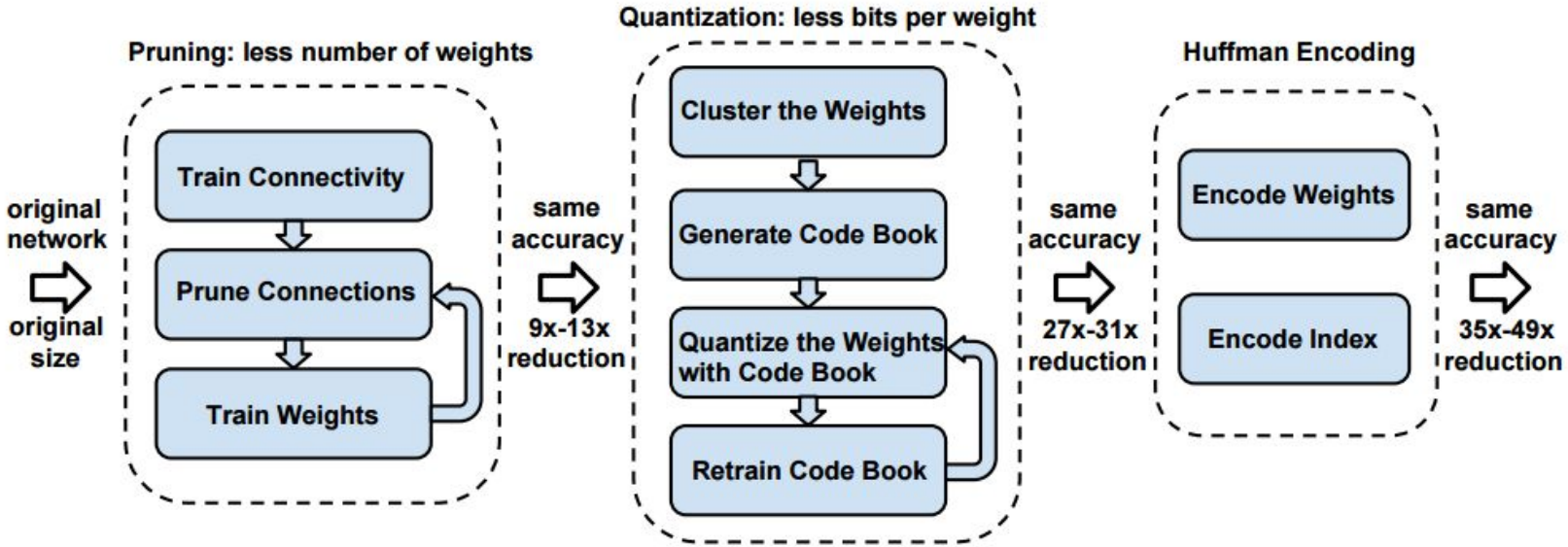


Figure 1: The three stage compression pipeline: pruning, quantization and Huffman coding. Pruning reduces the number of weights by $10\times$, while quantization further improves the compression rate: between $27\times$ and $31\times$. Huffman coding gives more compression: between $35\times$ and $49\times$.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

Overview

- Background

- System Design and Compression Pipeline

- Experiments

- Related/Future work and Conclusion

System Design and Compression Pipeline

NETWORK PRUNING

- prune small-weight connections below a threshold
- retrain the network for the remaining sparse connections
  - reduced # of parameters by 9× and 13× for AlexNet and VGG-16 model
- store the sparse structure using compressed sparse row/column format
  - requires 2a + n + 1 numbers, where a is # of non-zero elements and n is # of rows/columns
- store the index difference instead of the absolute position to compress further
  - 8 bits for conv layer and 5 bits for fc layer

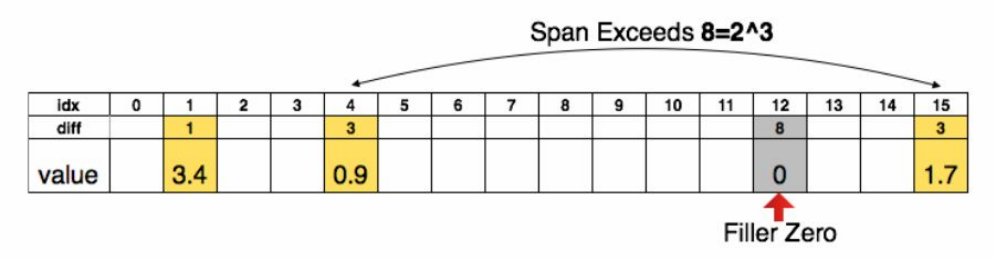


Figure 2: Representing the matrix sparsity with relative index. Padding filler zero to prevent overflow.

QUANTIZATION & WEIGHT SHARING

- limit # of distinctive weights by sharing weights between connections, and then fine-tune shared weights
  - compression rate for a network with n connections, each represented in b bits, with only k shared weights

$$r = \frac{nb}{nlog_2(k) + kb}$$

- k-clustering
  - To minimize the within-cluster sum of squares (WCSS)

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

- back-propagation

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

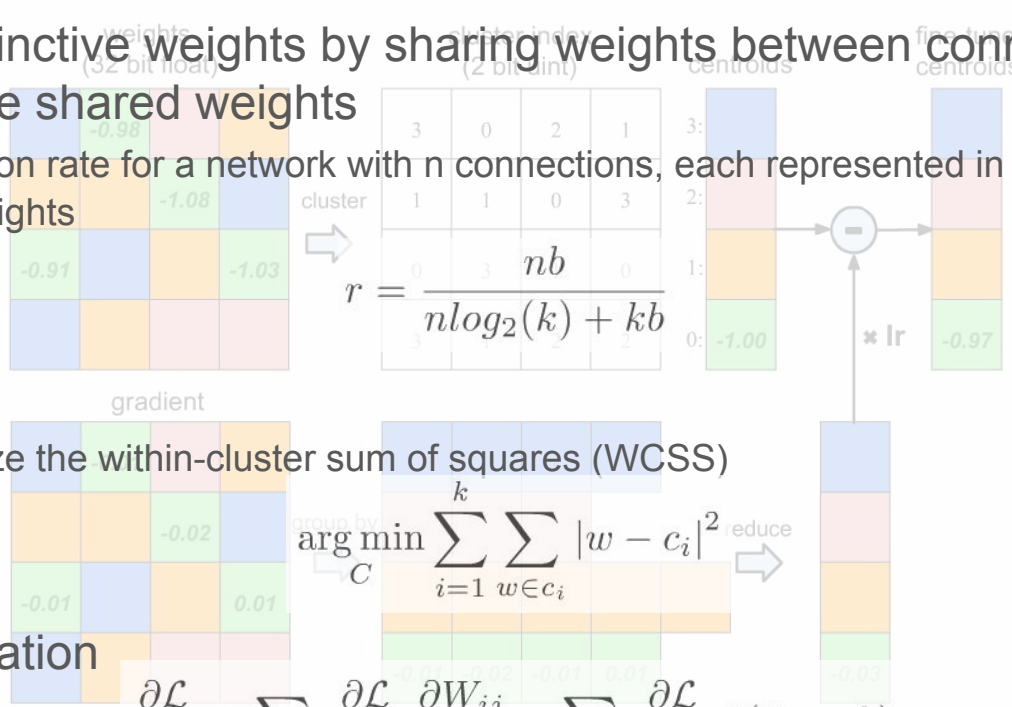


Figure 3: Weight sharing by scalar quantization (top). Trained centroids fine-tuning (bottom) for an example 4-input 4-output layer.

QUANTIZATION & WEIGHT SHARING

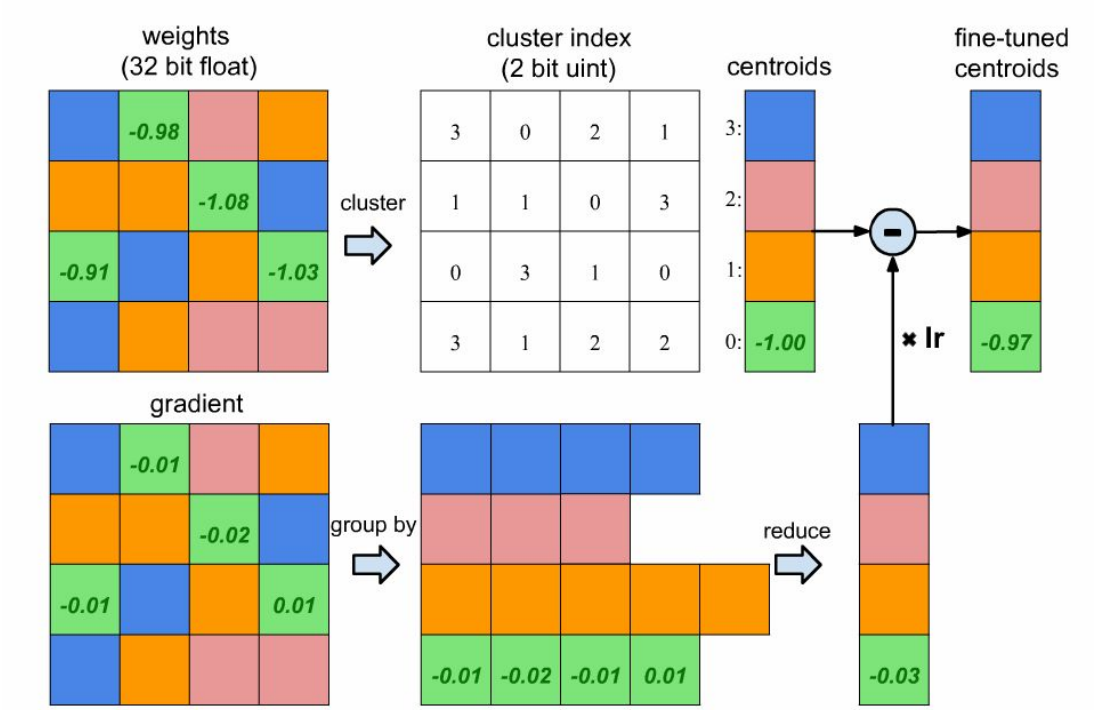


Figure 3: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom) for an example 4-input 4-output layer.

WEIGHT SHARING - Initialization

- Initialization of centroids
  - Forgy (random) initialization
    - Randomly chooses k observations from the data set
  - Density-based initialization
    - Pick points linearly spaced with respect to the CDF of the weights
  - Linear initialization
    - Pick points linearly spaced between the [min, max] of the weights
  - *Experiments shows that linear initialization works best*
    - *Larger weights play a more important role than smaller weights (Han et al., 2015)*

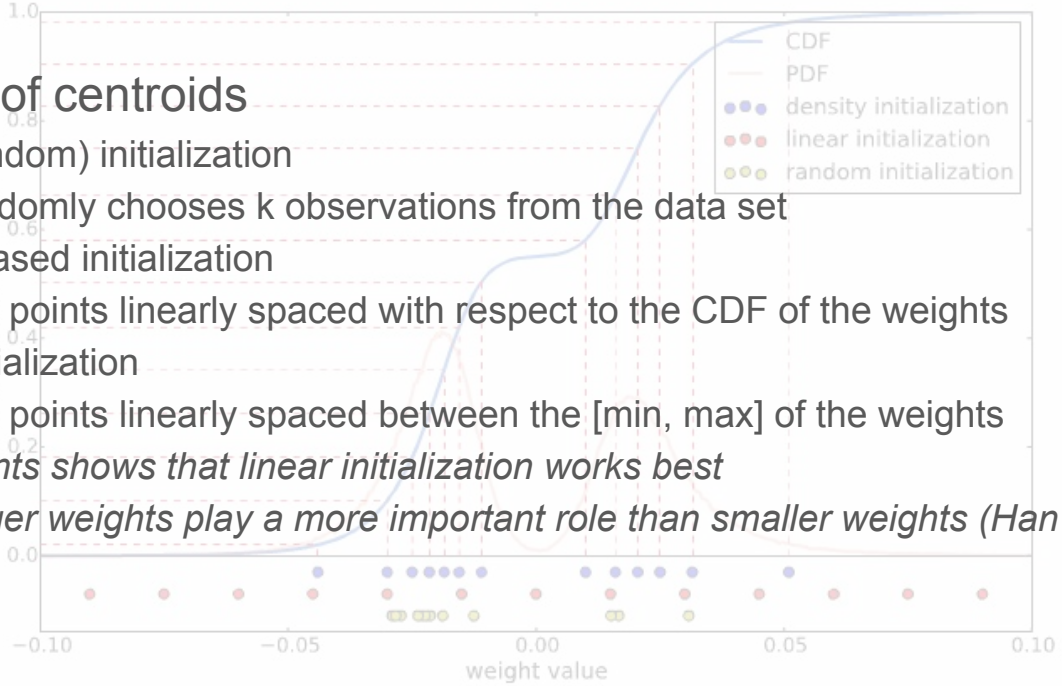


Figure 4 (left): Three different methods for centroids initialization.

WEIGHT SHARING - Initialization

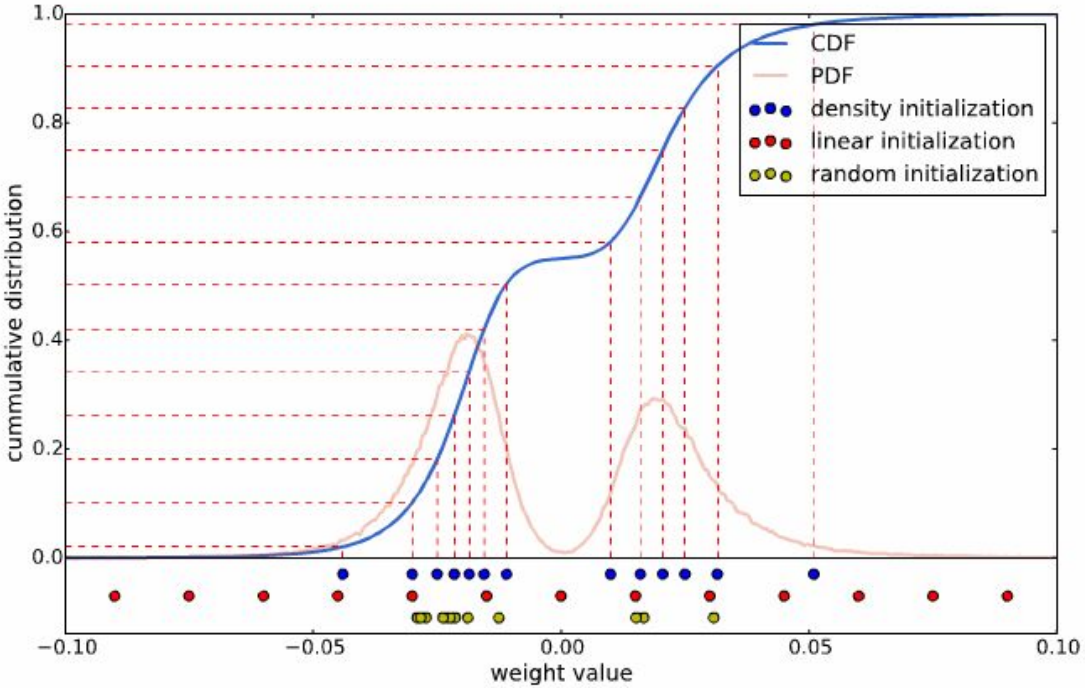


Figure 4 (left): Three different methods for centroids initialization.

WEIGHT SHARING - Initialization

- Initialization of centroids (cont'd)
  - *Experiments shows that linear initialization works best*
    - *Larger weights play a more important role than smaller weights (Han et al., 2015)*

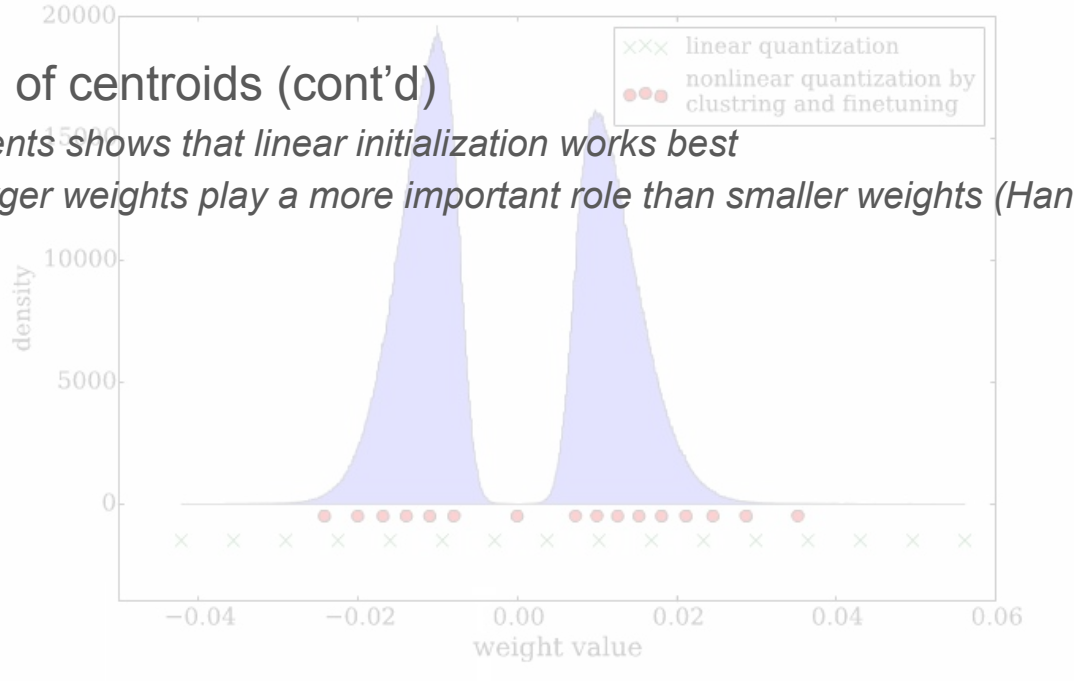


Figure 4 (right): Distribution of weights (blue) and distribution of codebook before (green cross) and after fine-tuning (red dot).

WEIGHT SHARING - Initialization

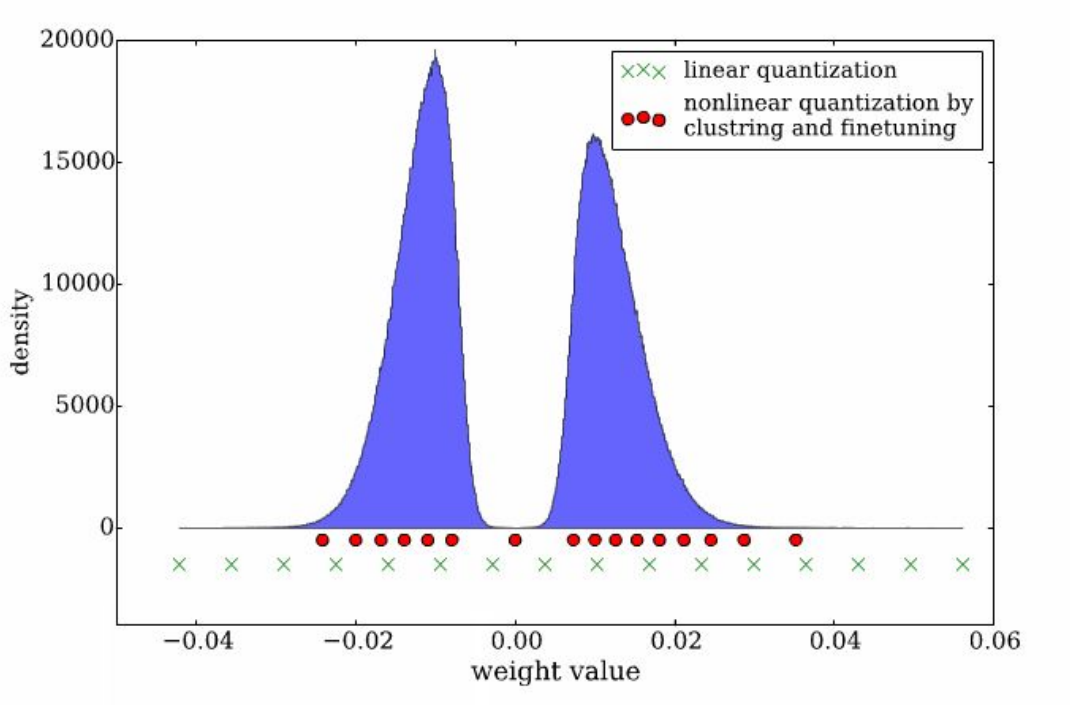
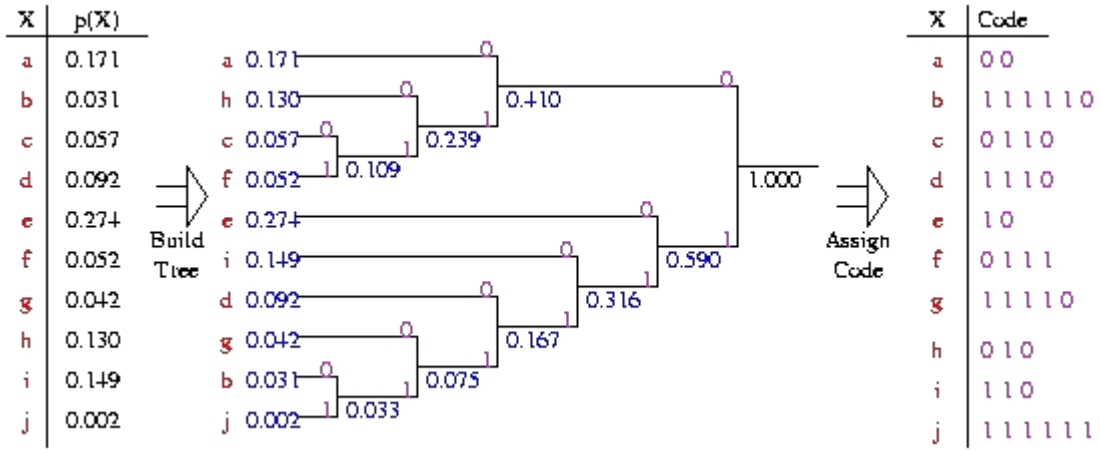


Figure 4 (right): Distribution of weights (blue) and distribution of codebook before (green cross) and after fine-tuning (red dot).

HUFFMAN CODING

- An optimal prefix code commonly used for lossless data compression (Van Leeuwen, 1976)
  - Experiments show this saves 20% − 30% of network storage



An illustration of generating Huffman coding

HUFFMAN CODING

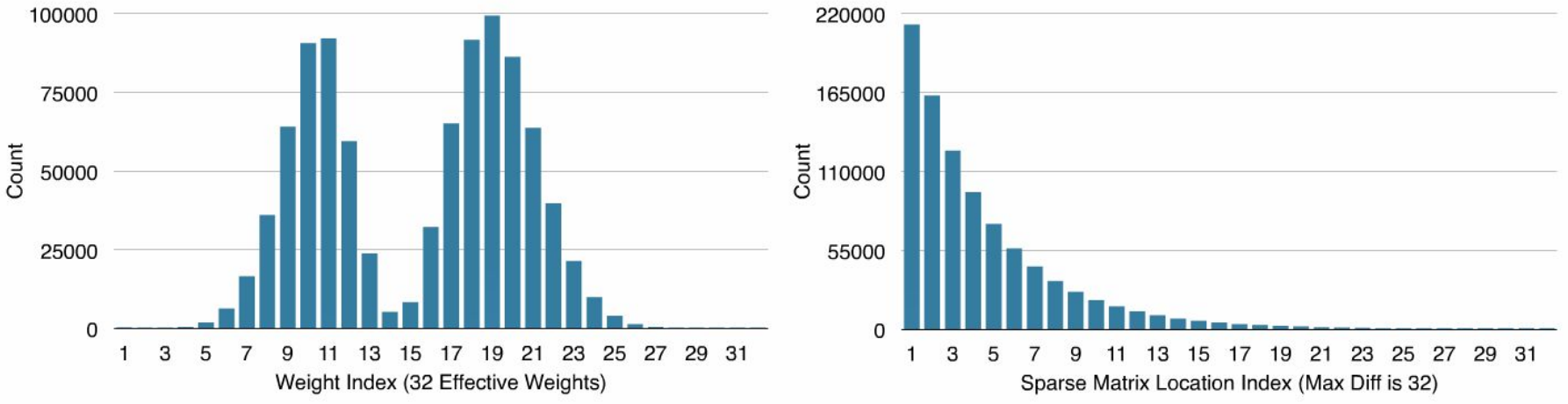


Figure 5: Distribution for weight (Left) and index (Right). The distribution is biased.

Experiments

Summary

- Datasets: MNIST (60,000+10,000), ImageNet (1.2M+50k)

- Neural Networks:
  - MNIST: LeNet-300-100 (266k, FC), LeNet-5 (431k, CONV)
  - ImageNet: AlexNet (61M, CONV), VGG-16(138M, CONV)

- Convolutional Layer (CONV), Fully Connected Layer (FC)

Overall Performance

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**×

Accuracy versus Compression Rate

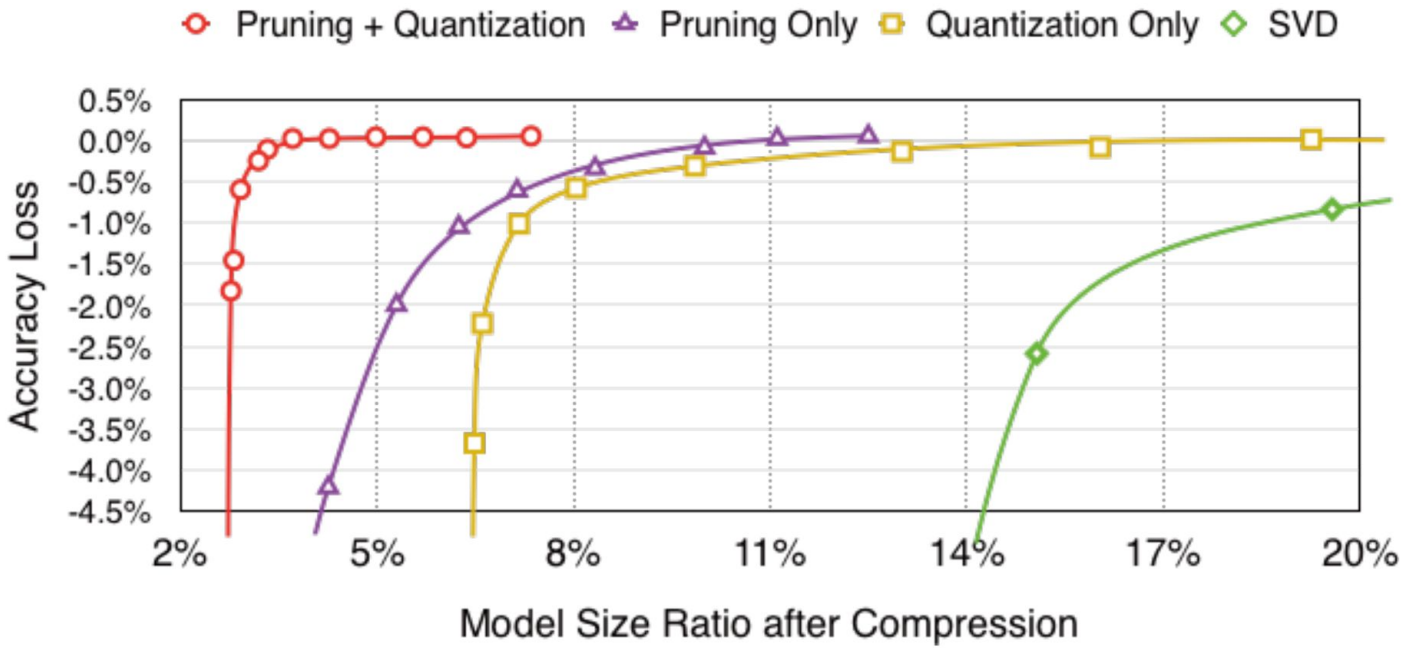


Figure 6: Accuracy v.s. compression rate under different compression methods. Pruning and quantization works best when combined.

Quantization

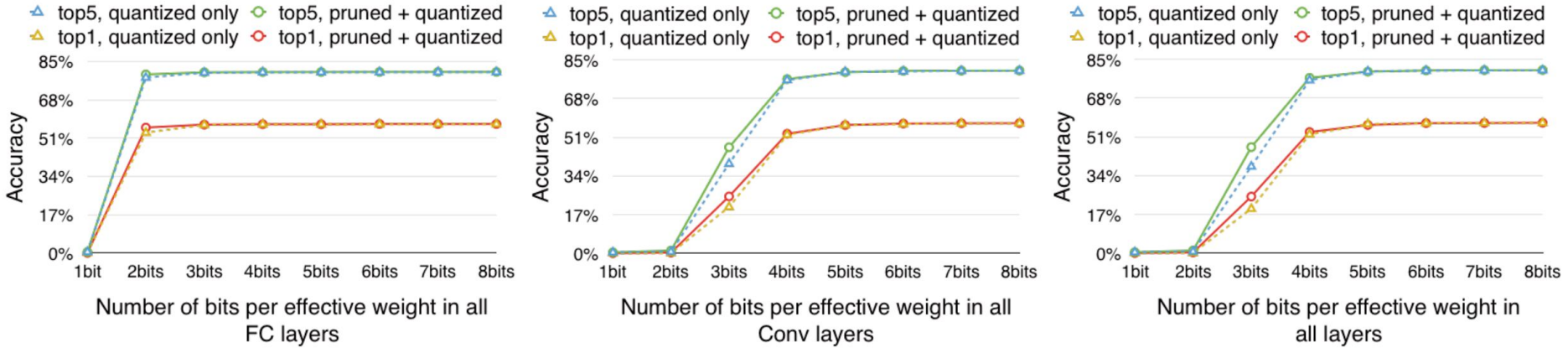


Figure 7: Pruning doesn't hurt quantization. Dashed: quantization on unpruned network. Solid: quantization on pruned network; Accuracy begins to drop at the same number of quantization bits whether or not the network has been pruned. Although pruning made the number of parameters less, quantization still works well, or even better(3 bits case on the left figure) as in the unpruned network.

Various Initialization Strategies

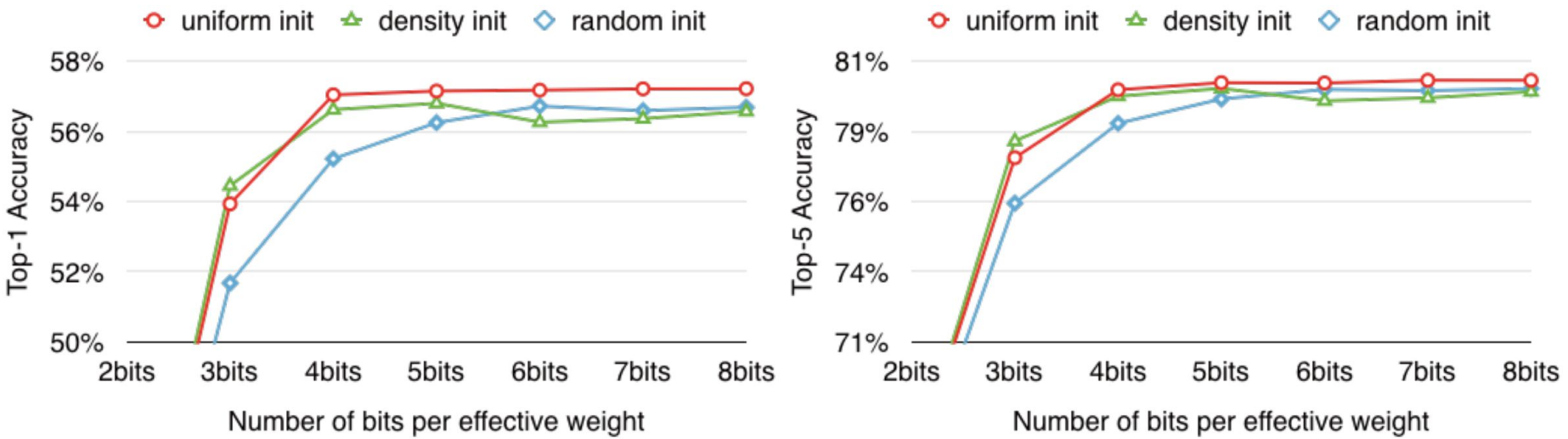


Figure 8: Accuracy of different initialization methods. Left: top-1 accuracy. Right: top-5 accuracy. Linear initialization gives best result.

Runtime

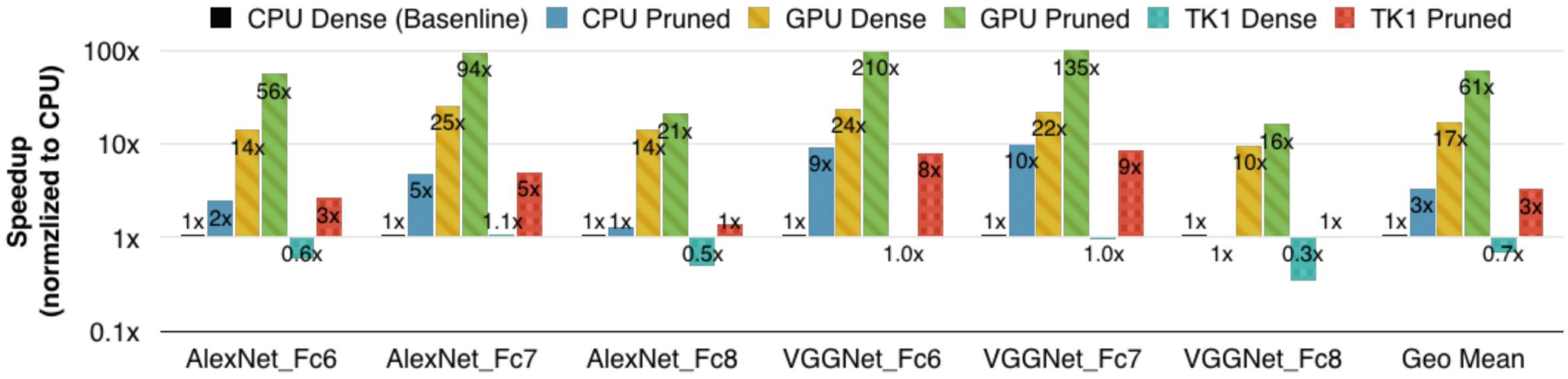


Figure 9: Compared with the original network, pruned network layer achieved $3\times$ speedup on CPU, $3.5\times$ on GPU and $4.2\times$ on mobile GPU on average. Batch size = 1 targeting real time processing. Performance number normalized to CPU.

Energy Efficiency

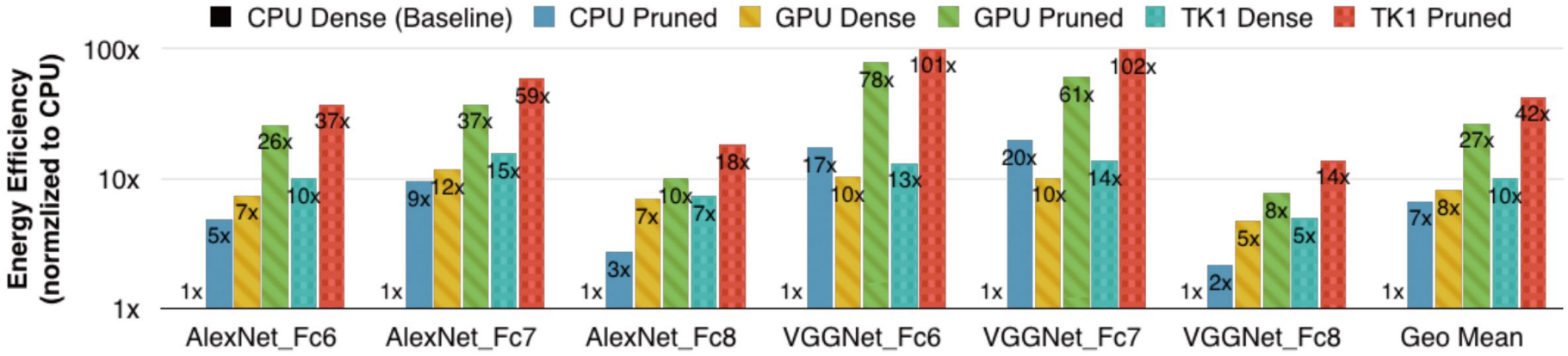


Figure 10: Compared with the original network, pruned network layer takes 7× less energy on CPU, 3.3× less on GPU and 4.2× less on mobile GPU on average. Batch size = 1 targeting real time processing. Energy number normalized to CPU.

Related/Future work and Conclusion

Related Work

- Data precision reduction
- Parameter binning
- Pruning

Related/Future work and Conclusion

What this paper focuses on:

- Pruning, Data precision reduction
- Parameter binning (weights clustering)
- Huffman Encoding

Future work:

- Theoretical Accuracy guarantee?
- Sparse Encoding?

The End

Any Questions?