EAI 2024 Fall

Paper Review Assignment 2

A Review of ”Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding”

學生：施宇庭 NN6124030

指導教授：蔡家齊 助理教授

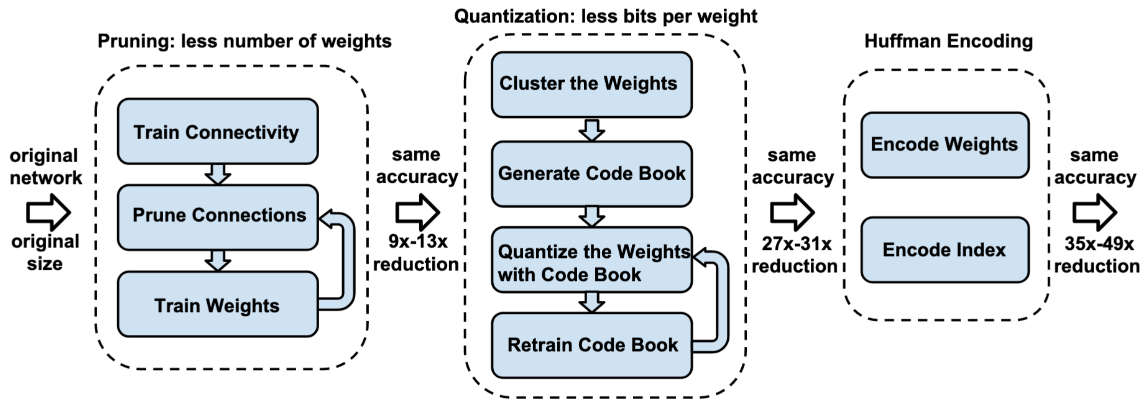

Figure 1: Overview of the three stages of Deep Compression pipeline: pruning, quantization, and Huffman coding [**han2015deep_compression**].

1 Motivation

The goal of this paper is to compress deep neural networks (DNNs) required to run inference with large models on edge/mobile devices. The motivation for this work is to address the following challenges:

- Reduce model size
- Reduce memory footprint
- Improve inference speed
- Reduce communication overhead
- Reduce energy consumption
- Maintain model accuracy

With the insight that pruning and quantization are able to compress the network without interfering each other, They proposed ”Deep Compression” combining pruning, quantization, and Huffman coding to keep accessing weights on chip instead of going to off-chip DRAM.

2 Proposed Solution

This paper proposes the ”Deep Compression” method, which combines pruning, quantization, and Huffman coding to compress neural networks to 1/35 to 1/49 of their original size while maintaining the same accuracy as the uncompressed network, as illustrated in Figure 1.

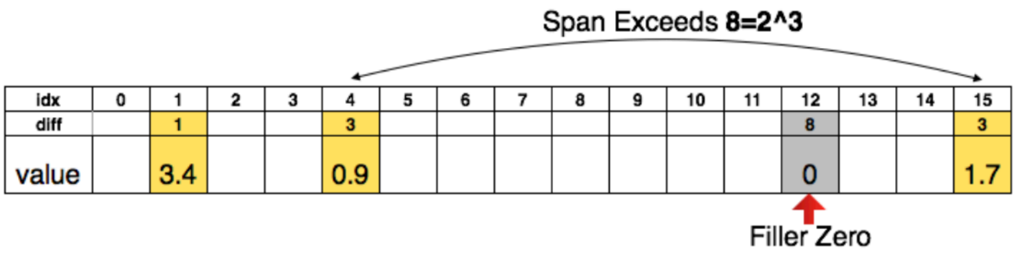

Figure 2: Representing the matrix sparsity with relative index. Padding filler zero to prevent overflow [**han2015deep_compression**].

2.1 Network Pruning

The pruning method adopts the approach proposed in the author's previous work **??**. The pruning process consists of three steps:

1. Learn the connectivity via normal network training
2. Remove the connections with small weights (below a threshold)
3. Iteratively fine-tune/retrain the network to recover the accuracy

The sparse matrix after pruning is stored as compressed sparse row (CSR) or compressed sparse column (CSC) format, which requires $2a + n + 1$ numbers, where a is the number of non-zero elements, and n is the number of rows or columns.

Note that they store the index difference instead of the absolute position, and both the values and indices will be quantized later (CONVs use 8 bits, and FCs use 5 bits). If the index difference of adjacent non-zero elements exceeds the quantization range, they will use a filler zero to prevent overflow, as shown in Figure 2.

2.2 Trained Quantization

As shown in Figure 1 and 3, the codebook-based quantization quantization with weight sharing consists of the following steps:

1. K-means clustering to find the weight value to be shared within the same cluster
2. Generate the codebook and quantize the weights
3. Iteratively Fine-tune the centroids in the codebook via gradient descent

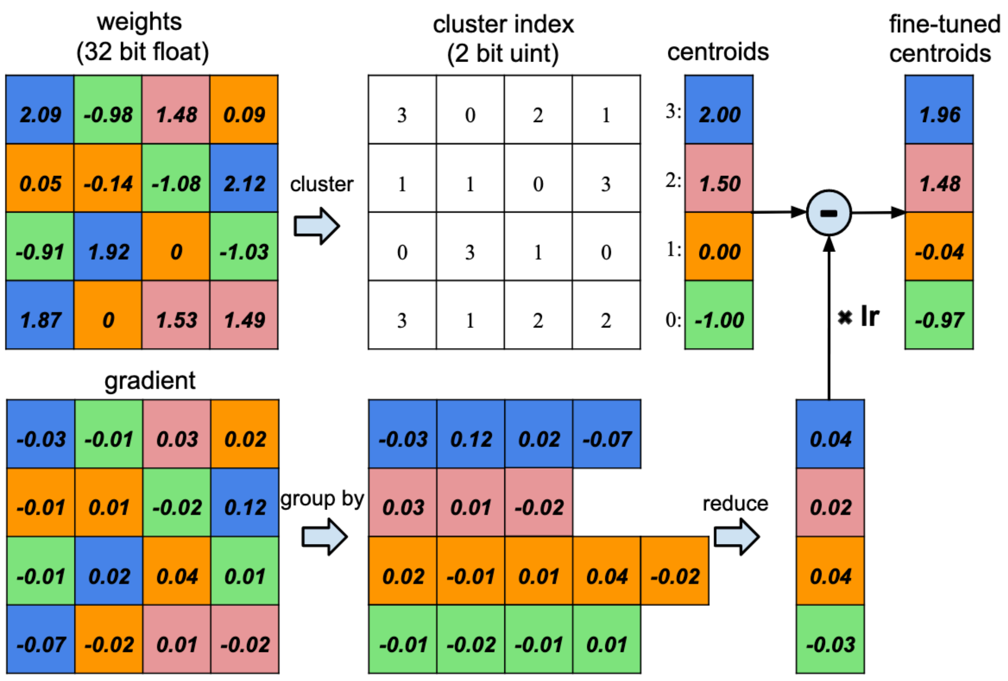

Figure 3: Trained quantization process: weight sharing by scalar quantization (top) and centroid fine-tuning (bottom) [**han2015deep_compression**].

Given k clusters, and only $\log_2(k)$ bits are used to encode the index. For a network with n connections and each connection is represented with b bits (e.g. $b = 32$ for FP32), the compression rate is:

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

For example, Figure 3 shows an FC layer with 4 inputs and 4 outputs, so the weights has $n = 4 \times 4 = 16$ elements. The number of clusters/centroids are $k = 4$, and the original floating-point numbers needs $b = 32$ bits, so the compression rate is $r = \frac{16\times32}{16\times2+4\times32} = 3.2$.

2.3 Huffman Coding

Huffman coding is a lossless data compression technique that assigns variable-length codes to the inputs based on their frequency of occurrence. The more frequent the input, the shorter the code. Figure 4 shows the distribution of weight (left) and sparse matrix index (right) of the last FC layer in AlexNet, which have large variance in their frequency, and it is therefore suitable for Huffman coding.

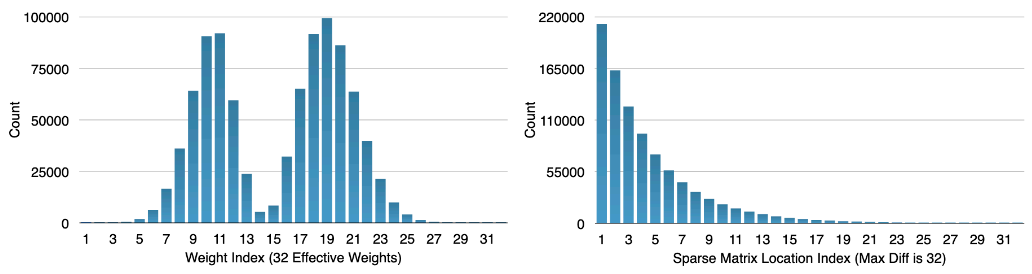

Figure 4: Distribution of weight (left) and sparse matrix index (right) [**han2015deep_compression**].

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**×

Figure 5: The top-1 and top-5 errors and compression rate before/after applying the proposed compression method.

3 Experiment and Evaluation

3.1 Image Classification Tasks

They evaluate the proposed method on the following image classification tasks, and the results are shown in Table 5. The compression statistics are shown in Table 6, 7, 8, and 9.

1. LeNet-300-100 on MNIST (Figure 6)
2. LeNet-5 on MNIST (Figure 7)
3. AlexNet on ImageNet (Figure 8)
4. VGG-15 on ImageNet (Figure 9)

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)
ip1	235K	8%	6	4.4	5	3.7	3.1%	2.32%
ip2	30K	9%	6	4.4	5	4.3	3.8%	3.04%
ip3	1K	26%	6	4.3	5	3.2	15.7%	12.70%
Total	266K	8%(12×)	6	5.1	5	3.7	3.1% (**32**×)	2.49% (**40**×)

Figure 6: LeNet-300-100 on MNIST.

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	0.5K	66%	8	7.2	5	1.5	78.5%	67.45%
conv2	25K	12%	8	7.2	5	3.9	6.0%	5.28%
ip1	400K	8%	5	4.5	5	4.5	2.7%	2.45%
ip2	5K	19%	5	5.2	5	3.7	6.9%	6.13%
Total	431K	8%(12×)	5.3	4.1	5	4.4	3.05% (**33**×)	2.55% (**39**×)

Figure 7: LeNet-5 on MNIST.

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**×)

Figure 8: AlexNet on ImageNet.

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**×)

Figure 9: VGG-16 on ImageNet.

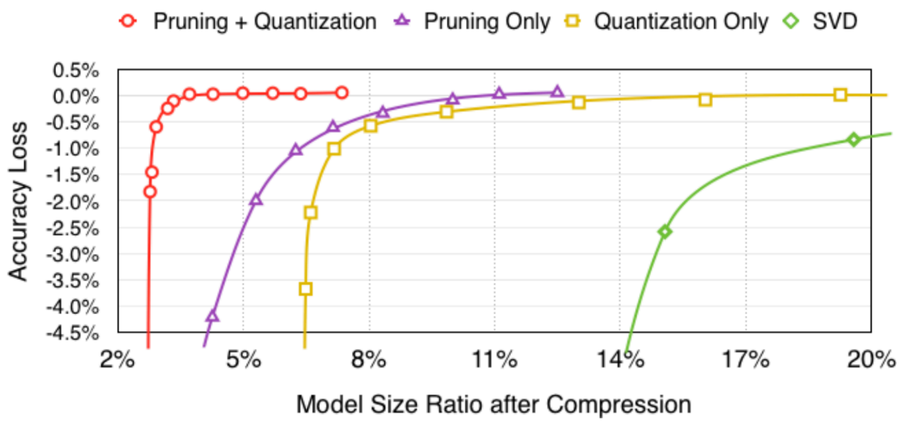

Figure 10: Accuracy v.s. compression rate under different compression methods. Pruning + quantization works the best.

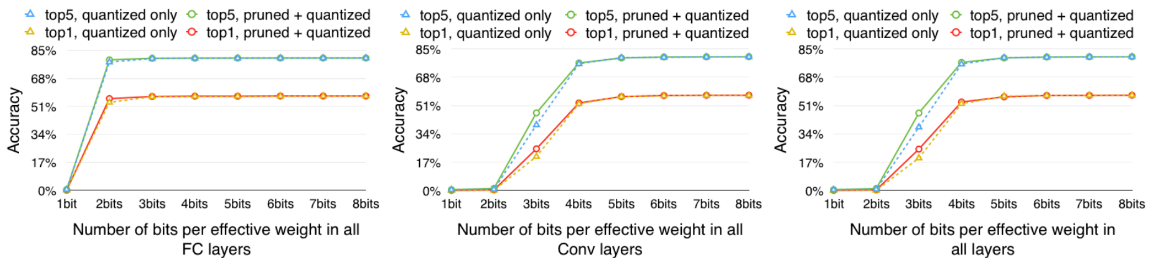

Figure 11: Accuracy v.s. compression rate under different compression methods. Pruning + quantization + Huffman coding works the best.

3.2 Pruning and Quantization Working Together

Figure 10 shows pruning combined with quantization (the leftest read line) works the best. Even being compressed to 3% of the original size, there is no loss of accuracy.

Figure 11 shows the accuracy between quantization only and pruning + quantization under different number of bits. For both CONVs and FCs, pruning works well with quantization.

3.3 Centroid Initialization

This paper have tried three different methods for centroids initialization (Figure 12):

1. Random initialization: randomly select k observations from the dataset
2. Density-based initialization: linearly spaces the centroids of the weights in the y-axis, then find the corresponding x-axis value.
3. Linear initialization: linearly spaces the centroids of the weights between [min, max] in the x-axis

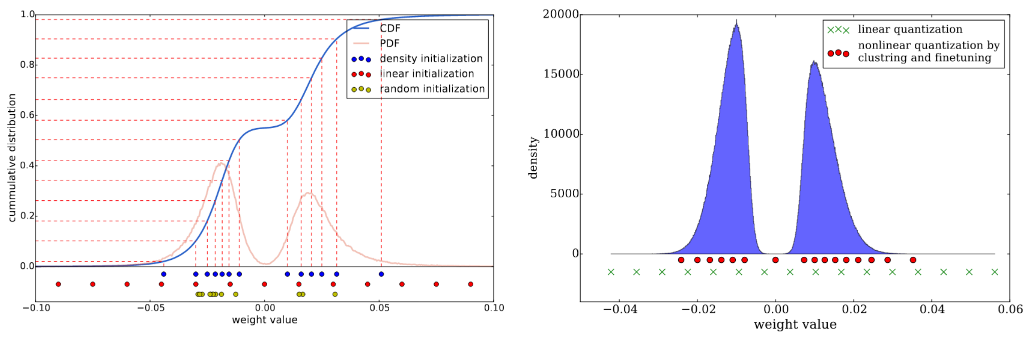

Figure 12: Left: Three different methods for centroids initialization. Right: Distribution of weights (blue) and distribution of codebook before (green cross) and after fine-tuning (red dot) [**han2015deep_compression**].

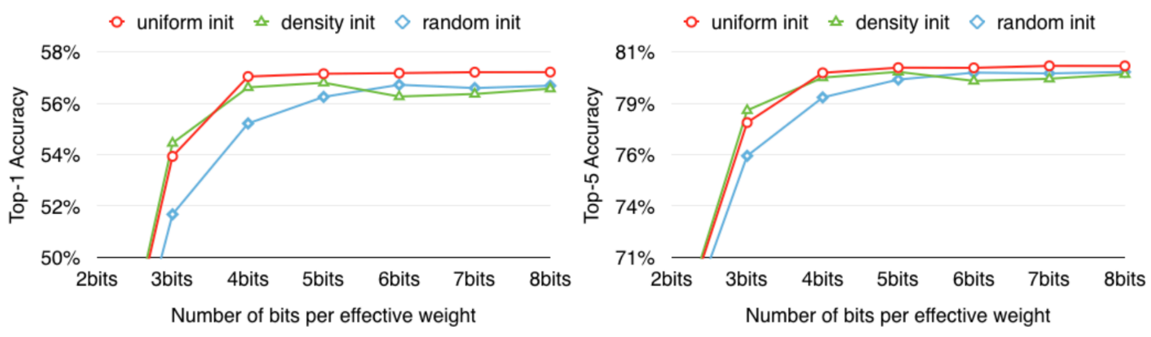

Figure 13: Three different methods for centroids initialization.

Figure 13 compares the accuracy of the three different initialization methods. Linear initialization outperforms the other two methods in all cases except at 3 bits.

3.4 Speedup and Energy Efficiency

Figure 14 illustrates the speedup of pruning on different hardware. When batch size is 1, the pruned network obtains 3x to 4x speedup over the dense network due to the smaller memory footprint and less data transferring overhead.

Figure 15 illustrates the energy efficiency of pruning on different hardware. When batch size is 1, the pruned network comsumes 3x to 7x less energy over the dense network.

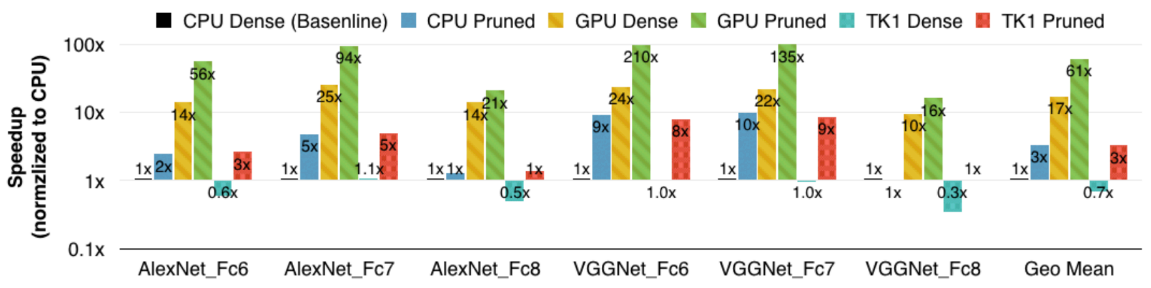

Figure 14: Speedup comparison on different hardware.

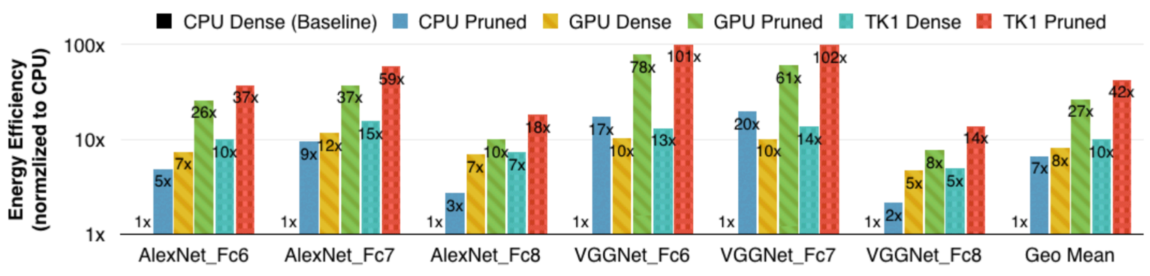

Figure 15: Energy efficiency comparison on different hardware.

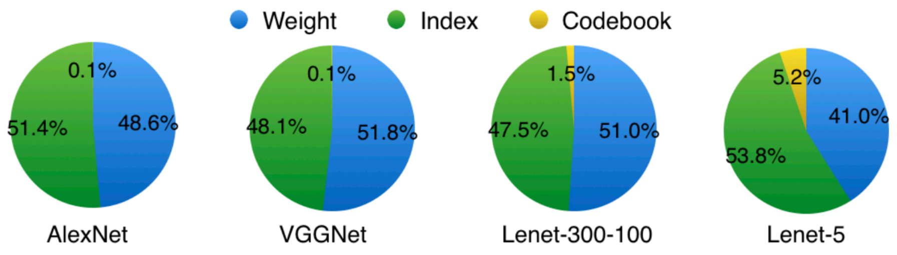

Figure 16: The storage ratio of weights, index, and codebook in the compressed model.

3.5 Ratio of Weights, Index, and Codebook

As Figure 16 shows, the storage of weights and the sparse indexes are roughly half and half respectively since on average they are encoded with 5 bits. The overhead of codebook is very small and often negligible.

4 Analysis

This paper has the following advantages and disadvantages:

4.1 Advantages

- **Proposed a systematic compression framework**
  Combines pruning, quantization, and encoding into a complete compression framework, forming an end-to-end process from model training to deployment, providing a clear methodology for model compression.

- **Multi-stage compression strategy**
  Sequentially applies pruning to reduce redundant connections, quantization to lower bit width, and Huffman coding to further compress storage requirements. Each step has a clear technical goal and complements the others. This strategy achieves a compression ratio of up to 35–49×, demonstrating significant effectiveness.

- **Practicality of pruning methods**

Uses importance-based weight pruning to effectively remove redundant parameters with minimal impact on model performance, significantly reducing model size. Fine-tuning the pruned model can restore its original performance.

- **Innovative quantization techniques**
  Introduces weight sharing and quantization, significantly reducing the number of bits needed by representing weights using cluster centers and index-based storage. Compared to traditional quantization techniques, this approach maintains model accuracy by leveraging learned cluster centers.

- **Comprehensive theoretical and experimental validation**
  Provides experimental results on multiple models (e.g., AlexNet and VGG-16), demonstrating the method's effectiveness across various deep learning applications. Quantitative analysis of the trade-off between compression ratio and model performance showcases its ability to achieve efficient compression while maintaining accuracy.

- **Strong generality**
  The framework is not limited to specific neural network architectures or application scenarios, exhibiting strong generality and applicability to various CNN models.

- **Reduces deployment cost**
  By compressing the model, it significantly reduces storage and transmission costs, facilitating the deployment of deep learning models on resource-constrained devices, such as mobile and embedded systems.

- **Decreases energy consumption and computation cost**
  Pruning and quantization not only reduce model size but also decrease the number of multiplication operations during inference, effectively lowering energy consumption and inference latency.

- **Compatibility with existing techniques**
  Techniques like Huffman coding can seamlessly integrate with existing compression methods, further improving compression efficiency with minimal impact on the training process.

4.2 Disadvantages

- **Weight sharing via k-means limitations**
  The k-means-based weight sharing approach forces unrelated weight elements to update together, as similar values are grouped, which may negatively impact training.

- **Quantization using codebooks**
  Although codebook quantization replaces most floating-point operations with integer op-

erations, floating-point computations are still required during both fine-tuning and inference.

- **Unstructured pruning's hardware unfriendly nature**
  The unstructured pruning approach is less hardware-friendly. The CSR or CSC compressed encoding used is not supported by all hardware. Software-based transformations (e.g., special kernel libraries) introduce additional overhead, and decompressing before computation negates the benefits of compression, such as reduced memory footprint, data movement, and off-chip DRAM accesses.

- **Huffman coding's inference overhead**
  Huffman coding introduces additional decoding operations during inference, potentially increasing computation overhead on general-purpose hardware like CPUs or GPUs. Whether the reduced memory access overhead compensates for the encoding/decoding overhead requires further analysis and comparison with other compression methods, such as weight decomposition or knowledge distillation.

- **Lack of support for ultra-low precision**
  The paper primarily explores higher-bit quantization (8-bit for CONVs and 5-bit for FCs), but does not address challenges of ultra-low precision. Modern Transformer models often adopt more aggressive low-precision formats (e.g., INT4 or lower) for efficient deployment.

- **Insufficient hardware-specific optimizations**
  The compressed models (via quantization and pruning) may not fully leverage the theoretical compression benefits on specific hardware (e.g., GPUs, TPUs). Hardware not optimized for sparse matrix operations offers limited performance gains.

- **Lack of structured pruning**
  Structured pruning (e.g., removing entire rows or blocks of parameters) is more suited for hardware acceleration. This paper focuses on unstructured pruning, which is less hardware-friendly.

- **Limited evaluation on modern architectures**
  At the time, architectures like ResNet, MobileNet, EfficientNet, and Transformer were not yet developed. The applicability of the proposed methods to these newer architectures requires further exploration.

5 Future Directions

- **Huffman Coding and Precision Optimization**: Huffman coding encodes quantized numbers based on their frequency, using fewer bits for common values and more bits for rare values. In current Transformer families, outliers often appear in activations, with most elements having similar magnitudes. This approach is similar to mixed precision quantization. A potential optimization could involve combining these techniques: applying Huffman coding on mixed-precision quantized numbers and accelerating through hardware implementation.

- **Limitations of K-means Weight Sharing**: The paper's k-means weight sharing approach forces unrelated weight elements to update together, despite similar values not necessarily indicating synchronous updates. A better approach would involve considering gradient dynamics during clustering and redesigning the WSCC loss function to more accurately capture weight element relationships.
- **Structured Pruning**: Extend pruning from unstructured to structured approaches, targeting attention heads, Transformer layers, or entire modules to better leverage hardware characteristics like sparse matrix multiplication acceleration.
- **Quantization Optimization**: Support ultra-low precision quantization (e.g., INT4 or binarization) combined with mixed-precision techniques, preserving precision for critical components while reducing bit width for others.
- **Sparsity Optimization and Dynamic Sparsity**: Leverage the inherent sparsity in modern LLM attention mechanisms through learned sparsity methods like sparse attention and long-range sparse mechanisms to further reduce computational complexity.
- **Multimodal Architecture-Specific Compression**: Design specialized compression techniques for different modalities in multimodal models, such as CNN-inspired methods for image modalities and language knowledge distillation for text modalities.
- **Combining Distillation and Compression**: Utilize knowledge distillation and deep compression as complementary techniques, transferring knowledge from large Transformer models to smaller, compressed models while integrating pruning and quantization.
- **End-to-End Optimization**: Develop end-to-end training processes specifically for compression techniques, rather than applying them post-training, to minimize performance impact from pruning and quantization.