Deep k-Means: Re-Training and Parameter Sharing with Harder Cluster Assignments for Compressing Deep Convolutions

Junru Wu [1] **Yue Wang** [2] **Zhenyu Wu** [1] **Zhangyang Wang** [1] **Ashok Veeraraghavan** [2] **Yingyan Lin** [2]

Abstract

The current trend of pushing CNNs deeper with convolutions has created a pressing demand to achieve higher compression gains on CNNs where convolutions dominate the computation and parameter amount (e.g., GoogLeNet, ResNet and Wide ResNet). Further, the high energy consumption of convolutions limits its deployment on mobile devices. To this end, we proposed a simple yet effective scheme for compressing convolutions though applying k-means clustering on the weights, compression is achieved through weight-sharing, by only recording K cluster centers and weight assignment indexes. We then introduced a novel spectrally relaxed k-means regularization, which tends to make hard assignments of convolutional layer weights to K learned cluster centers during re-training. We additionally propose an improved set of metrics to estimate energy consumption of CNN hardware implementations, whose estimation results are verified to be consistent with previously proposed energy estimation tool extrapolated from actual hardware measurements. We finally evaluated Deep k-Means across several CNN models in terms of both compression ratio and energy consumption reduction, observing promising results without incurring accuracy loss. The code is available at https://github.com/Sandbox3aster/Deep-K-Means

1. Introduction

Convolutional neural networks (CNNs) have gained considerable interest due to their record-breaking performance in many recognition tasks (Krizhevsky et al., 2012; Girshick et al., 2013; Taigman et al., 2014). In parallel, there has been a tremendously growing need to bring CNNs into resource-constrained mobile devices in line with the recent surge of edge computing in which raw data are processed locally in edge devices using their embedded machine learning algorithms (Shi et al., 2016) (Lin et al., 2017). The advantage lies in that local processing avoids transferring data back and forth between data centers and edge devices, thus reducing communication cost, latency, and enhancing privacy. However, deploying CNNs into resource-constrained platforms is a non-trivial task. Devices at the edge, such as smart phones and wearables, have limited energy, computation and storage resources since they are battery-powered and have a small form factor. In contrast, powerful CNNs require a large number of weights that corresponds to considerable storage and memory bandwidth. For example, the amount of weights in state-of-the-art CNNs AlexNet and VGG-16 are over 200MB and 500MB, respectively (Han et al., 2016). Further, CNN-based applications can drain a battery very quickly if executed frequently. For example, smartphones nowadays cannot even run classification using AlexNet in real-time for more than one hour (Yang et al., 2017).

To close the gap between the constrained resources of edge devices and the growing complexity of CNNs, compression techniques have been widely investigated to reduce the precision of weights and the number of operations during or after CNN training in order to shrink their large implementation cost while maintaining the desired inference performance. Various CNN compression techniques have been proposed, such as weight compression (Bhattacharya & Lane, 2016) (Han et al., 2016) (Lane et al., 2016) and decomposition (Changpinyo et al., 2017) (Howard et al., 2017) (Wang et al., 2015), and compact architectures (Iandola et al., 2016) (Lin et al., 2014). However, there are two major shortcomings in existing CNN compression techniques.

- A myriad of CNN compression techniques focus on the fully-connected layers of CNNs which convention-

[1] Department of Computer Science and Engineering, Texas A&M University, College Station, TX, USA; [2] Department of Electrical and Computer Engineering, Rice University, Houston, TX, USA. Correspondence to: Zhangyang Wang <atlaswang@tamu.edu>, Yingyan Lin <yingyan.lin@rice.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

ally have dominant parameters. However, recent successful CNNs tend to shift more parameters towards convolutional layers and have only one or even no fully-connected layers. For example, 85% of the parameters lie in the convolutional layers of GoogleNet (Chen et al., 2016a). Despite the growing trend toward CNN models using more convolutional layers and fewer fully-connected layers, only few compression techniques are dedicated for convolutional layers.

- The majority of CNN compression techniques, including most of the very few that focus on compressing convolutional layers, are designed to merely reduce the CNN model size or the amount of computation, which does not necessarily lead to reduced energy consumption. In fact, the recent work (Yang et al., 2017) argues that the number of weights, multiply-and-accumulate (MAC) operations, and speedup ratio are often not good approximations for energy consumption, which also heavily depends on memory data movement and more. For example, the authors show an interesting result that although SqueezeNet (Iandola et al., 2016) has 51.8× fewer weights than AlexNet, it consumes 33% more energy due to its larger amount of computation and data movement. A compression technique that aims to reduce both model size and energy-aware complexity is hence highly desired for enabling extensive resource-constrained CNN applications.

1.1. Contribution

In this paper, we propose *Deep k-Means*, a compression pipeline that is well suited for trimming down the complexity of convolutional layers that dominate both the model size as well as energy consumption of recently developed state-of-the-art CNNs. *Deep k-Means* consists of two steps. First, a novel spectrally relaxed k-means regularization is developed to enforce highly clustered weight structures during re-training. After that, compression is performed via weight-sharing, by only recording cluster centers and weight assignment indexes. We evaluate the performance of *Deep k-Means* in comparison with several state-of-the-art compression techniques focused on compressing convolutional layers. The results show that *Deep k-Means* consistently achieves higher accuracy at the same compression ratio (CR) as its competitors. Furthermore, *Deep k-Means* is also evaluated in terms of energy-aware metrics developed by us, and its compressed models show favorable energy efficiency as well. Our main contributions are summarized as follows:

- We introduce a novel spectrally relaxed k-means regularization that automatically learns hard(er) assignments of convolutional layer weights during re-training, to favor the subsequent compression via k-means weight-sharing. Our regularization approach is effective, efficient, simple to implement and use, and easily scalable to large CNN models.
- Inspired by a recently developed dataflow called "row-stationary", that minimizes data movement energy consumption on CNN hardware implementation (Chen et al., 2016b), we reformulate the weights into row vectors for weigh-sharing clustering. Such a formulation has the potential to result in CNN models that are in favor of energy-efficient hardware implementation.
- In order to bridge the gap between algorithm and hardware design of CNNs, we propose an improved set of energy-aware metrics based on (Sakr et al., 2017). Our energy consumption estimation results are verified to be consistent with those from the tool in (Yang et al., 2017), which was extrapolated from actual hardware measurements. We expect our metrics to broadly benefit future research in energy-aware CNN design.

1.2. Related Work

Parameter pruning and sharing has been used both to reduce network complexity and to avoid over-fitting. Early pruning approaches include Biased Weight Decay (Hanson & Pratt, 1989), Optimal Brain Damage (Cun et al., 1990), and Optimal Brain Surgeon (Hassibi & Stork, 1993). Recent works (Srinivas & Babu, 2015) made use of the redundancy among neurons. The Deep Compression method introduced in (Han et al., 2016) employed a three stage pipeline to prune the redundant connections, quantize the weights via scalar weight sharing, and then encode the quantized weights using Huffman coding. An effective soft weight-sharing method described in (Ullrich et al., 2017) showed competitive CRs on state-of-the-art CNNs, e.g., Wide ResNet.

With fully-connected layers traditionally considered as the memory bottleneck, numerous works focused on compressing these layers. For example, (Gong et al., 2014b) proposed applying k-means clustering to the densely-connected layers and showed a good balance between model size and accuracy. (Chen et al., 2015) proposed HashedNet that used a low-cost hash function to group weights into hash buckets for parameter sharing. On the other hand, a few recent works embraced the trend towards more convolutional layers in CNNs and attempted to compress convolutional layers. For example, (Chen et al., 2016a) proposed an architecture called FreshNets to compress filters of convolutional layers in the frequency domain. The recent work (Abbasi-Asl & Yu, 2017) iteratively pruned filters based on the classification accuracy reduction index, and achieved substantially higher classification accuracy compared to other structural compression schemes, e.g., (He et al., 2014; Li et al., 2016).

2. Proposed Approach

2.1. Parameter Sharing via Row-wise k-Means

Assuming a convolutional layer $\in \mathbb{R}^{s \times s \times c \times m}$, where s denotes the filter size, c the input channel number, and

m the output channel number. Following the convention in CNNs, we reshape it as a matrix $W \in \mathbb{R}^{s \times N}$, where $N = s \times c \times m$, each column vector $\in \mathbb{R}^s$ being <u>a row</u> from an original convolutional filter. Following the product quantization approach for fully-connected layers in (Gong et al., 2014a), we treat all columns of W as N samples, and apply k-means to assign them with K clusters. When $K \ll N$, we need only to store the cluster indexes and codebooks after k-means. We define the *cluster rate* for each layer as $\frac{K}{N}$ here. For compressing multiple convolutional layers, we adopt a "uniform parameter sharing" scheme for simplicity, i.e., each convolutional layer chooses its K value such that all layers have the same cluster rate, expect for the first layer whose cluster rate is often set higher.

We notice other alternatives to enforce structured parameter sharing among convolutional layers using k-means, e.g., reshaping each convolutional filter as vectors R^{s^2} and then clustering over $c \times m$ samples, or converting each output channel into $\mathbb{R}^{s \times s \times c}$ and clustering over resulting m samples. In practice, we find their performance to be close (with k chosen in different proper ways). One major motivation for choosing the row-wise k-means is that it could lead to higher data reuse opportunity and thus result in more energy-efficient hardware implementations, according to the row-stationary dataflow recently proposed in (Chen et al., 2016b), which has shown to be superior in terms of energy efficiency compared to other dataflows. Another motivation arises from reducing the complexity (see Section 2.2).

2.2. k-Means Regularized Re-Training

Simply pruning or sharing weights in CNNs will usually hurt the inference accuracy. Re-training has often been exploited to enforce the favorable structures in the pruned/shared weights and compensate for the accuracy loss (Han et al., 2016). In order to be compatible with k-means parameter sharing, we would favor a re-training scheme that "naturally" encourages the weights to be concentrated tightly around, or exactly at, a number of cluster components which are optimized for high predictive accuracy. The goal is fulfilled by introducing a novel *spectrally relaxed k-Means regularization* below.

The original sum-of-squares function of k-Means usually employs a Lloyd-type algorithm to solve. The spectral relaxation technique of k-Means was introduced in (Zha et al., 2002), by first equivalently re-formulating sum-of-squares into a trace form with special constraints. Specifically, to cluster N samples of $\mathbb{R}^s$, represented as $W \in \mathbb{R}^{s \times N}$, into K clusters, the spectral relaxation converts the k-means objective into the following problem:

$$\min_{W; F \in \mathcal{F}} Tr(W^T W) - Tr(F^T W^T W F), \tag{1}$$

where Tr denotes the matrix trace. $F \in \mathbb{R}^{N \times k}$ is the normalized cluster index matrix, and $\mathcal{F}$ denotes its special structure requirement: $F_{ij} = 1/\sqrt{n_j}$ if column i belongs to the cluster j and there is a total of n_j samples in the cluster j; and $F_{ij} = 0$ otherwise, $i = 1, ..., N$, $j = 1, ..., K$, and $\sum_{j=1}^{K} n_j = N$. The original spectral relaxation (Zha et al., 2002) considers W as given; thus (1) is reduced to:

$$\max_{F \in \mathcal{F}} Tr(F^T W^T W F) \tag{2}$$

The authors of (Zha et al., 2002) then proposed ignoring the special structure of F and let it be an arbitrary orthogonal matrix. (2) is thus relaxed to a trace maximization problem over a Stiefel manifold:

$$\max_{F} Tr(F^T W^T W F),\ s.t.\ F^T F = I \tag{3}$$

It results in a closed-form solution of F, by composing the first k singular vectors of W, according to the well-known Ky Fan theorem.

As a <u>critical difference</u> with (Zha et al., 2002), here our goal is not to cluster a *static* W. Rather, we would like to encourage W to stay "suited" for k-means during the dynamic re-training, without incurring a significant increase in complexity. We are thus motivated to utilize (1) as a regularization term on learning W, rather than a stand-alone objective. We discuss just one convolutional layer W for simplicity: assume that the original CNN training minimizes the energy function $E(W)$, w.r.t. W. The retraining minimizes the regularized objective below (λ is a scalar):

$$\begin{aligned} &\min_{W,F} E(W) + \frac{\lambda}{2}[Tr(W^T W) - Tr(F^T W^T W F)], \\ &s.t.\ F^T F = I \end{aligned} \tag{4}$$

Note that F is treated as an auxiliary variable to promote a clustered structure in W. Solving (4) could be iterated between the updates of W and F. Updating W can follow the standard stochastic gradient descent (SGD), with the gradient given as: $\nabla E(W) + \lambda W(I - FF^T)$. F is updated using the same closed-form solution to (3), by computing the k-truncated singular value decomposition (SVD) of W.

By the interaction between F and W during re-training, the regularization keeps W in a highly clustered state, in addition to optimizing it for inference accuracy. Although F has been relaxed from the "hard" normalized cluster index matrix to an arbitrary orthogonal one, we observe in practice that it still tends to enforce weights close to those taking around K unique vector values, i.e., encouraging "approximately hard" (or "harder" than soft weight sharing) K-cluster assignments during re-training.

In *Deep k-Means*, starting from an uncompressed pre-trained model as initialization, we will re-train it with adding this novel data-dependent weight regularizer (4) to each convolutional layer, while other training protocols remain unchanged. The re-training typically converges into a much smaller number of epochs than in the original training. After that, we apply row-wise k-means on the learned W for the final parameter-sharing step.

Complexity Analysis For each convolutional layer W, the extra complexity incurred by applying the spectrally relaxed k-means regularization term includes two parts: (1) updating W: the only extra burden is to compute $\lambda W(I - FF^T)$, which takes $\mathcal{O}(sKN)$ or $\mathcal{O}(s^2cmK)$ (computing WFF^T); (2) updating F via SVD, which costs $\mathcal{O}(s^2N)$ or $\mathcal{O}(s^3cm)$: that also serves another motivation to create W with lower row dimensions (e.g., s rather than s^2 or s^2c), since it will reduce the SVD complexity of W. Considering that s is usually small, the total extra complexity $\mathcal{O}((s^2K + s^3)cm)$ is quite affordable, enabling our methods to scale well for modern CNNs. In practice, we also implement the F update in a very "lazy" way so that SVD will merely be computed once for every five epochs, for further accelerating the re-training, with only marginal impacts on the result.

2.3. Comparison with Existing Work

Directly enforcing a k-means friendly weight structure is not straightforward. (Ullrich et al., 2017) presents an elegant and inspiring Bayesian regularization form of "soft cluster assignment". During re-training, the authors fit a Gaussian mixture model (GMM) prior model over the weights, to encourage the distribution of weights to be close to K clusters. After re-training, each weight was quantized to the mean of the GMM component that takes most responsibility, for parameter sharing. Their pipeline is the closet peer work to ours, with the major difference being that we pursue harder cluster assignment during re-training. As is well known, GMM is reduced to k-means when the mixture variance gets close to zero. Therefore, the retraining process in (Ullrich et al., 2017) could also be viewed as a "softened" version of k-means. However, the differences between the two methods manifest in multiple folds:

- First, our "harder" cluster assignment is directly derived from the original k-means objective (1). We expect it to be better aligned with the k-means parameter sharing stage. Our experimental observations show that this leads to more skewed weight distributions, and achieves better results than (Ullrich et al., 2017).
- Second, compared to the Bayesian form in (Ullrich et al., 2017), our regularization adds very little extra complexity to the standard SGD. The implementation only calls for minor changes (a new regularizer term); and thanks to its low complexity, it is ready to be applied to larger-scale CNNs.
- Third, (Ullrich et al., 2017) discussed their high sensitivity to the choices of learning rates for mixture parameters (e.g., means, log-variances): a higher learning rate may cause model collapse and a lower one results in slow convergence. In contrast, *Deep k-Means* has merely one hyper-parameter λ. We find *Deep k-Means* insensitive to λ (λ between 10^{-4} and 10^{-3} is found to work almost equally well). *Deep k-Means* needs no special learning rate scheduling. It is also free of postprocessing, e.g., removing redundant components as (Ullrich et al., 2017) needed to.

3. Energy-Aware Metrics for CNN Energy Consumption Estimation

While CR or reduction in the number of operations are widely adopted by existing CNN compression techniques as generic performance metrics, these metrics are not necessarily tied to improved energy efficiency as pinpointed by (Yang et al., 2017) according to their energy estimation tool extrapolated from actual hardware measurements. Therefore, it is important to evaluate compression techniques using a set of energy-aware metrics other than CR.

However, it is non-trivial to estimate the energy consumption of CNNs because a significant portion of energy consumption in CNNs is consumed by data movement, which mainly depends on the employed memory hierarchy and dataflow when implementing CNNs and is thus difficult to be estimated directly from the model. An energy estimation tool extrapolated from actual hardware measurements was proposed by (Yang et al., 2017) to bridge the gap between algorithm and hardware design. Unfortunately, their tool currently only supports AlexNet and GoogLeNet_v1.

We hereby propose the following energy-aware metrics:

- **Computational cost** measures the computational resources needed to generate a single decision and is defined in terms of the number of 1 bit full adders (FAs), which is a canonical building block of arithmetic units. Specifically, assuming that the arithmetic operations are executed using the commonly employed ripple carry adder and BaughWooley multiplier architectures, the number of FAs needed to compute a D-dimensional dot product between the activations and weights is (Lin et al., 2016):

$$DB_{\mathbf{w}}B_{\mathbf{x}} + (D-1)(B_{\mathbf{x}} + B_{\mathbf{w}} + \lceil \log_2(D) \rceil - 1) \quad (5)$$

  where $B_{\mathbf{w}}$ and $B_{\mathbf{x}}$ denote the fixed-point precision assigned to the weights and activations, respectively.

- **Weight representational cost** measures the storage complexity and data movement costs corresponding to the weights and is defined as the product between the total number of bits needed to represent all weight parameters and the total number of times that the weights are used to compute convolutions:

$$N_{\mathbf{w}} \left| \mathcal{W} \right| B_{\mathbf{w}} \quad (6)$$

where $N_{\mathbf{w}}$ and $\mathcal{W}$ denote the total number of times that the weights are used to compute convolutions and the index sets of all weights in the network, respectively.

- **Activation representational cost** is similar to the weight representational cost above and is defined as:

$$N_{\mathbf{x}} \, |\mathcal{X}| \, B_{\mathbf{x}} \tag{7}$$

$N_{\mathbf{x}}$ and $\mathcal{X}$ denote the total number of times that the activations are used to compute convolutions and the index sets of all activations in the network, respectively.

The concepts of computational and representational costs were first proposed in (Sakr et al., 2017) to describe network complexity. To better reflect the CNN energy cost, we modify the definition of representational cost in (Sakr et al., 2017) to include the number of times that weights or activations are loaded for computing convolutions, in order to capture the associated data movement cost. Specifically, if a certain weight filter is removed due to compression, then the corresponding activation would be loaded less frequently, thus leading to reduced data movement costs. This can be reflected by the reduction of $N_{\mathbf{x}}$ in our modified definition but not the originally defined representational cost in (Sakr et al., 2017). Also, we separate the representational costs for the weights and activations to evaluate the impact of compression in more detail.

4. Experiments

We evaluate *Deep k-Means* in terms of CR and energy-aware metrics respectively, with the resulting accuracy loss Δ after compression, using two sets of experiments. The default λ is 10^{-4}. Unless otherwise specified, we will focus on compressing convolutional layers only.

For the first set of experiments on CR, we first create a simple baseline CNN for simulation experiments w.r.t. varying CR. The CR definition follows (Han et al., 2016). We then compare *Deep k-Means* against four latest and competitive comparison baselines for compressing convolutional layers: Ultimate Tensorization (Garipov et al., 2016), FreshNet (Chen et al., 2016a), Greedy Filter Pruning (Abbasi-Asl & Yu, 2017), and Soft Weight-Sharing (Ullrich et al., 2017). The first three have been optimized towards compressing CNNs dominated by the convolutions and reported results on their self-designed models. (Ullrich et al., 2017) outperformed strong baselines such as (Han et al., 2015) on the standard MNIST benchmark; the authors then reported compression results on the state-of-the-art Wide ResNet model (Zagoruyko & Komodakis, 2016) that mainly consist of convolutions. We also compare *Deep k-Means* with the baseline of *Deep k-Means* without re-training (*Deep k-Means WR*), i.e., directly performing row-wise k-means on original weights.

For the second set of experiments, we first validate the estimated energy consumption using our metrics to match the actual hardware-based extrapolation (Yang et al., 2017), and then evaluate *Deep k-Means* against the aforementioned baselines from an energy consumption perspective. While it is overall challenging to estimate energy consumption accurately due to the multitude of factors involved, our proposed metrics are simple, effective (i.e., showing a good match with the results extrapolated by actual hardware measurement), and thus can help CNN model designers understand various design trade-offs. We also provide insights regarding the impact of different types (i.e., parameter-sharing or pruning) of compression techniques on the computational and representational costs in Eqs. (5), (6) and (7).

4.1. Comparison on Compression Ratio

4.1.1. Comparison with Ultimate Tensorization

Model	Δ (%)	CR
TT-conv (naive)	-2.4	2.02
TT-conv (naive)	-3.1	2.90
TT-conv	-0.8	2.02
TT-conv	-1.5	2.53
TT-conv	-1.4	3.23
TT-conv	-2.0	4.02
Deep k-Means	+0.05	2
Deep k-Means	-0.04	4

Table 1. Compressing TT-conv-CNN in (Garipov et al., 2016).

.

(Garipov et al., 2016) proposed a tensor factorization based method specifically for compressing convolutional layers. The authors proposed a Tensor Train (TT) Decomposition approach for convolutional kernel, denoted as *TT-conv (naive)*. It could be further enhanced by introducing a new type of TT-conv layer, denoted as *TT-conv*. The authors evaluated TT-conv (naive) and TT-conv on a self-designed architecture, called TT-conv-CNN, consisting of six convolutional layers and one fully-connected layer. TT-conv-CNN is dominated by the convolutions (occupying 99.54% parameters of the network), and the authors reported the uncompressed model's top-1 accuracy of 90.7% on CIFAR-10 (Krizhevsky & Hinton, 2009).

We evaluate *Deep k-Means* on the TT-conv-CNN model at CR = 2 and 4. Table 1 compares them with the compression results in (Garipov et al., 2016). *Deep k-means* incurs minimal accuracy loss even at CR = 4. More surprisingly, it even slightly increases the accuracy after compression at CR = 2. It concurs with the previous observations by (Ullrich et al., 2017; Cheng et al., 2017): removing parameter redundancy improves CNN generalization on some small networks.

4.1.2. Comparison with FreshNet

The Frequency-Sensitive Hashed Nets (FreshNets) was proposed in (Chen et al., 2016a) to exploit inherent redundancy in convolutional layers. The authors observed that convolutional weights to be typically smooth and low-frequency. They were thus motivated to first convert filter weights to the frequency domain, after which they group frequency parameters into hash buckets to achieve parameter sharing. The authors evaluated their method on their self-designed CNN (referred to as *FreshNet-CNN* hereinafter) consisting of five convolutional layers and one fully-connected layer. They reported the uncompressed FreshNet-CNN to obtain the top-1 accuracy of 85.09% on CIFAR-10.

Table 2 reports the compression results of *Deep k-Means* and *Deep k-Means WR* on FreshNet-CNN at CR = 16. We also include two original baselines in (Chen et al., 2016a): low-rank decomposition (LRD) (Denil et al., 2013) and HashedNet (Chen et al., 2015). In this example, even the accuracy of *Deep k-Means WR* is very competitive. After retraining, *Deep k-Means* shows a sharp further improvement.

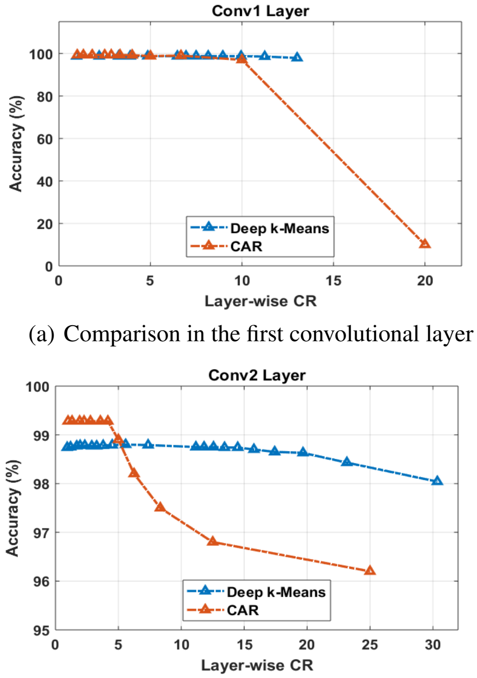
Model	Δ (%)	CR
LRD	-8.32	16
HashedNet	-9.79	16
FreshNet	-6.51	16
Deep k-Means WR	-5.95	16
Deep k-Means	-1.30	16

Table 2. Compressing FreshNet-CNN in (Chen et al., 2016a).

4.1.3. Comparison with Greedy Filter Pruning

The recent work (Abbasi-Asl & Yu, 2017) introduced a greedy structural compression scheme that prunes redundant convolutional filters in a trained CNN, based on a classification accuracy reduction (CAR) algorithm. The authors reported promising results on LeNet-5, AlexNet and ResNet-50, and their evaluations adopted a unique layer-wise compression fashion: taking LeNet-5 for example, each time the authors pruned filters in one convolutional layer (first or second) while leaving other layers untouched, and then reported the overall accuracy.

We compare *Deep k-Means* with CAR (with re-training, the best performer in (Abbasi-Asl & Yu, 2017)) using LeNet-5 on MNIST, and follow their layer-wise compression setting. Thus, unlike our other experiments, we report the accuracy w.r.t. "layer-wise" CR, i.e., measuring how many times the current layer is compressed, rather than the overall CR that measures the entire model. As Figure 1 shows, both *Deep k-Means* and CAR produce similar results at small layer-wise CRs; CAR is more competitive at small CRs for Conv2. However, *Deep k-Means* is clearly superior at high layer-wise CRs for both layers.



(a) Comparison in the first convolutional layer

(b) Comparison in the second convolutional layer

Figure 1. Compressing LeNet following the layer-wise setting in (Abbasi-Asl & Yu, 2017): (a) The overall classification accuracy of LeNet when only the first convolutional layer (Conv1) is compressed, w.r.t. layer-wise CR; (b) The overall classification accuracy of LeNet when only the second convolutional layer (Conv2) is compressed, w.r.t. layer-wise CR.

4.1.4. Comparison with Soft Weight Sharing

Model	Δ (%)	CR
Soft Weight-Sharing	-2.02	45
Deep k-Means WR	-16.02	45
Deep k-Means WR	-25.45	47
Deep k-Means WR	-45.08	50
Deep k-Means	-1.63	45
Deep k-Means	-2.23	47
Deep k-Means	-4.49	50

Table 3. Compressing Wide ResNet in comparison to soft weight-sharing (Ullrich et al., 2017).

(Ullrich et al., 2017) reported the compression performance of soft weight-sharing on the state-of-the-art Wide ResNet model (Zagoruyko & Komodakis, 2016), a convolution-dominant CNN with 2.7M parameters, at one single CR = 45 using CIFAR-10 (the uncompressed baseline top-1 error is 6.48%). Thanks to the light computational burden of *Deep k-Means*, we are able to evaluate various CRs. Note that at the same CR, soft weight-sharing and *Deep k-Means* will lead to identical layer-wise dimensions and the same number of unique weights in each layer. Thus, their performance difference can only arise from the effects of their different regularization ways during re-training.

Promoting Sparsity in Re-Training. During the review stage, *one anonymous reviewer* commented that (Ullrich et al., 2017) tried to explicitly enforce weight values to a cluster centered at zero, while the above default routine of *Deep k-Means* had not such constraint. Such a sparsity-promotion operation may marginally decrease compression performance as it restricts the flexibility of setting centroids, but can gain more in both speedup and energy savings (Parashar et al., 2017). To ensure a fair comparison with (Ullrich et al., 2017), we implement a similar sparsity-promoting feature for *Deep k-Means*, *in this specific experiment only*. Without referring to sophisticated options such as semi-supervised clustering (Basu et al., 2002), we follow a simple heuristic which incurs almost no extra complexity: at each time of "lazy update" for layer $W \in \mathbb{R}^{s\times N}$, we first rank all N columns of W in terms of their ℓ_2 norms. We then assign the pN ($0 < p < 1$) smallest-norm columns to one cluster with a fixed center at zero, before solving (4). At the parameter-sharing step, we similarly threshold pN smallest-norm columns in W to be all-zero, and then perform $(k-1)$-clustering for remaining columns. The group of layer-wise p that we used for all 16 layers is: [0, 0.3, 0.4, 0.5, 0.4, 0.4, 0.5, 0.5, 0.5, 0.5, 0.5, 0.6, 0.9, 0.5, 0.75, 0.9].

Table 3 demonstrates the superiority of *Deep k-Means* (with the above-described sparsity promotion) over (Ullrich et al., 2017), by comparing their top-1 accuracy drops: 1.63% versus 2.02 %, at CR = 45. We further display the results at CR = 47 and 50, with a smooth accuracy decrease.

4.1.5. Evaluation with GoogleNet on ImageNet

We finally evaluate *Deep k-Means* on the GoogleNet (Szegedy et al., 2015) trained with the ImageNet ILSVRC12 dataset (Russakovsky et al., 2015). We use single center crop during testing, and evaluate the performance based on the *top-1* and *top-5* accuracy drops on the validation set, compared to the uncompressed baseline whose *top-1* accuracy is 69.76% and *top-5* 89.63%. We include two comparison methods: one-shot network compression (Kim et al., 2015), and low-rank regularization (Tai et al., 2015). According to Table 4, *Deep k-Means* proves to scale well on large models/datasets, and achieves significantly better results over the two baselines: its compression at CR ≤ 3 is almost lossless, with top-5 errors again observed to slightly increase after compression. The GoogleNet compression performance is found to deteriorate quickly when CR > 4.

Model	$\Delta^{\dagger}$ %	$\Delta^{\ddagger}$ %	CR
One-shot (Kim et al., 2015)	N/A	-0.24	1.28
Low-rank (Tai et al., 2015)	N/A	-0.42	2.84
Deep k-Means WR	-1.22	-0.65	1.5
Deep k-Means WR	-3.7	-2.46	2
Deep k-Means WR	-13.72	-10.05	3
Deep k-Means WR	-48.95	-48.82	4
Deep k-Means	-0.26	0.00	1.5
Deep k-Means	-0.17	+0.06	2
Deep k-Means	-0.36	+0.03	3
Deep k-Means	-1.95	-1.14	4

Table 4. Compressing GoogLeNet on ILSVRC12 († and ‡ are top-1 and top-5 accuracies respectively).

4.2. Comparison on Energy-Aware Metrics

4.2.1. Energy-Aware Metrics Verification

We first evaluate our energy-aware metrics by comparing its estimated energy consumption with that of the tool in (Yang et al., 2017). Note that the unit of energy: 1) in (Yang et al., 2017) is normalized in terms of number of MAC operations while the computational cost in Eq. (5) is normalized in terms of number of FAs; and 2) for the representational cost in Eqs. (6) and (7) is different from that of the computational cost in Eq. (5). Therefore, we first normalize the representational cost in terms of the computational cost assuming that a global on-chip buffer is employed, implying that the representational cost of a MAC is about 6 times that of performing a MAC computation (Chen et al., 2016b). This normalized representational cost is then added to the computational cost to obtain our total energy. Lastly, we normalize this total energy in terms of the number of MACs to be the same as that of (Chen et al., 2016b).

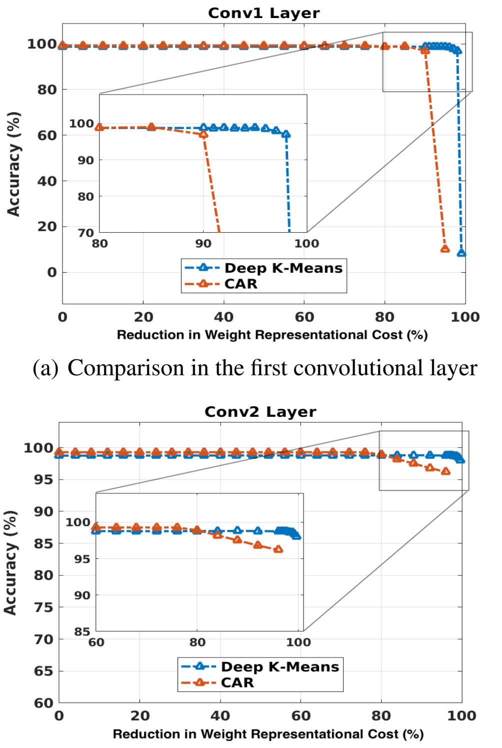
We calculate the coefficient of determination (R^2), between the estimated energy consumptions using our proposed metrics, and using the tool in (Yang et al., 2017), of the same compressed models. We use *Deep k-means* to compress both AlexNet and GoogLeNet_v1 [1], which are the *only two CNN models* currently supported by (Yang et al., 2017). The energy consumptions are estimated as we choose the *cluster rate* to vary between: (AlexNet) [0.5, 0.3, 0.25, 0.2, 0.15, 0.1, 0.05, 0.01], and (GoogLeNet_v1) [0.33, 0.18, 0.05, 0.012], respectively, to ensure negligible accuracy loss. We have R^2 to be **0.9931** for AlexNet, and **0.9675** for GoogLeNet_v1, suggesting the estimated energy consumptions using our proposed metrics to be *strongly linearly correlated* with the results extrapolated from actual hardware measurements (Yang et al., 2017). Yet different from their tool, our metrics are generally applicable to any CNN.

4.2.2. Comparison with Greedy Filter Pruning

An ideal CNN model to be deployed on resource-constrained platforms should simultaneously possess compact model size and low energy cost. In general, the result-

[1] For both networks, we employ our proposed methods in convolutional layers only. For AlexNet, we only quantize weight and activation to 8 bit and to 16 bit in fully-connected layers, respectively. For GooLeNet_v1, we use the one with global average pooling, which has no fully-connected layer.

ing computational/representational cost reduction via compression depends on how the network is trimmed, i.e., which parts of the network is compressed. Conceptually, we point out that different compression schemes (e.g., parameter-sharing versus pruning) will affect the analysis of the computational and representational costs defined in Eqs. (5), (6) and (7). First, the weight representational cost is directly proportional to CR in both parameter-sharing (e.g. *Deep k-Means* and soft weight-sharing) and pruning (e.g. CAR) cases, because they both in effect can reduce the term $|\mathcal{W}|$ in (6). Second, the activation representational cost is proportional to CR for the case of weight pruning, but is independent of CR if the compression is done by weight sharing. This is because weight pruning results in skipping of the corresponding computations and thus can reduce the number of times that the corresponding activations are used (i.e., $\mathcal{W}$ in (7)), whereas there is no computation or connection skipping in the case of weight-sharing. Third, the computational cost is again proportional to CR for weight pruning; yet it would become input-dependent when it comes to weight sharing. Specifically, only when all the weights corresponding to the same input/activation are shared, the computational cost reduction ratio becomes equal to CR.



(a) Comparison in the first convolutional layer

(b) Comparison in the second convolutional layer

Figure 2. Comparison between *Deep k-Means* and CAR, in terms of the ratio between weight representational cost reduction.

Deep k-Means has constantly obtained the best CR performance among the aforementioned baselines. To provide a concrete example, we choose the CAR (with re-training) baseline in (Abbasi-Asl & Yu, 2017), which produces slightly inferior but still competitive CR results, and discuss its potential energy efficiency improvement compared with *Deep k-Means*. The same layer-wise compression setting in Section 4.1.3 is adopted, for the first two convolutional layers of LeNet-5. A similar analysis could be done for other methods too.

Deep k-Means compresses the network via weight sharing, whereas CAR relies on weight pruning. The accuracy versus CR comparison (i.e., Figure 1) in Section 4.1.3 shows that *Deep k-Means* is clearly superior to CAR at high layer-wise CRs, for either of the two convolutional layers. The potential energy consumption comparison between *Deep k-Means* and CAR in terms of the three metrics are as follows[2].

First, *Deep k-Means* will achieve higher weight representational cost reduction since it mostly offers higher CR with the same or even better accuracy, in particular at high CRs. Figure 2 (a) and (b) compares the accuracy versus weight representational cost reduction ratio of *Deep k-Means* and CAR for compressing the first and second convolutional layers in LeNet 5, respectively[3]. We observe that *Deep k-Means* achieves about 10% and 17.2% higher weight representational cost reduction, respectively, compared to CAR when compressing the first and second layers, without incurring accuracy loss. Second, CAR always outperforms *Deep k-Means* in terms of activation representational cost, because it removes filters and thus reduces the numbers of feature maps. In theory, CAR can achieve up to (layer-wise) "CR times" better activation representational cost reduction ratio than *Deep k-Means*. Third, the achievable computational cost reduction by *Deep k-Means* is either smaller or equal to that of CAR, depending on the inputs.

5. Conclusion and Discussions

This paper proposes *Deep k-Means*, a retraining-then-parameter-sharing pipeline for compressing convolutional layers in deep CNNs. A novel spectrally relaxed k-means regularization is derived to make hard assignments of convolutional layer weights to learned cluster centers during re-training. *Deep k-Means* demonstrates clear superiority over several recently-proposed competitive methods, in terms of both compression ratio and energy efficiency. Our future work will exploit more adaptive cluster rates for different layers instead of the current uniform scheme. Based on our proposed metrics, we also aim to incorporate more energy-aware regularizations into *Deep k-Means* for direct minimization of energy consumptions.

[2] We are unable to directly verify the total energy consumption of CAR using our metrics due to the lack of their model parameters or pre-trained model.

[3] We did not consider the cost of weight assignment indexes as it is negligible due to achievable high CR.

Acknowledgements

We would like to thank all anonymous reviewers for their tremendously useful comments to help improve our work. We acknowledge the inspiring discussions with Dr. Yang Zhang at IBM Watson and Dr. Edwin Park at Qualcomm Technologies, Inc. We acknowledge the Texas A&M and Rice High Performance Research Computing for providing a part of the computing resources used in this research.

References

Abbasi-Asl, Reza and Yu, Bin. Structural compression of convolutional neural networks based on greedy filter pruning. *arXiv preprint arXiv:1705.07356*, 2017.

Basu, Sugato, Banerjee, Arindam, and Mooney, Raymond. Semi-supervised clustering by seeding. In *In Proceedings of 19th International Conference on Machine Learning (ICML-2002*. Citeseer, 2002.

Bhattacharya, Sourav and Lane, Nicholas D. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of SenSys*, 2016.

Changpinyo, Soravit, Sandler, Mark, and Zhmoginov, Andrey. The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257*, 2017.

Chen, Wenlin, Wilson, James T., Tyree, Stephen, Weinberger, Kilian Q., and Chen, Yixin. Compressing neural networks with the hashing trick. *CoRR*, abs/1504.04788, 2015.

Chen, Wenlin, Wilson, James, Tyree, Stephen, Weinberger, Kilian Q, and Chen, Yixin. Compressing convolutional neural networks in the frequency domain. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1475–1484. ACM, 2016a.

Chen, Y. H., Emer, J., and Sze, V. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 367–379, June 2016b.

Cheng, Yu, Wang, Duo, Zhou, Pan, and Zhang, Tao. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

Cun, Yann Le, Denker, John S., and Solla, Sara A. Advances in neural information processing systems 2. chapter Optimal Brain Damage, pp. 598–605. 1990. ISBN 1-55860-100-7.

Denil, Misha, Shakibi, Babak, Dinh, Laurent, De Freitas, Nando, et al. Predicting parameters in deep learning. In *Advances in neural information processing systems*, pp. 2148–2156, 2013.

Garipov, Timur, Podoprikhin, Dmitry, Novikov, Alexander, and Vetrov, Dmitry. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

Girshick, Ross B., Donahue, Jeff, Darrell, Trevor, and Malik, Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. volume abs/1311.2524, 2013.

Gong, Yunchao, Liu, Liu, Yang, Ming, and Bourdev, Lubomir. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014a.

Gong, Yunchao, Liu, Liu, Yang, Ming, and Bourdev, Lubomir D. Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115, 2014b.

Han, Song, Pool, Jeff, Tran, John, and Dally, William J. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015.

Han, Song, Mao, Huizi, and Dally, William J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proceedings of ICLR*, 2016.

Hanson, Stephen Jose and Pratt, Lorien Y. Comparing biases for minimal network construction with back-propagation. In Touretzky, D. S. (ed.), *Advances in Neural Information Processing Systems 1*, pp. 177–185. Morgan-Kaufmann, 1989.

Hassibi, Babak and Stork, David G. Second order derivatives for network pruning: Optimal brain surgeon. In Hanson, S. J., Cowan, J. D., and Giles, C. L. (eds.), *Advances in Neural Information Processing Systems 5*, pp. 164–171. Morgan-Kaufmann, 1993.

He, T., Fan, Y., Qian, Y., Tan, T., and Yu, K. Reshaping deep neural network for fast decoding by node-pruning. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 245–249, May 2014.

Howard, Andrew G, Zhu, Menglong, Chen, Bo, Kalenichenko, Dmitry, Wang, Weijun, Weyand, Tobias, Andreetto, Marco, and Adam, Hartwig. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Iandola, Forrest N, Han, Song, Moskewicz, Matthew W, Ashraf, Khalid, Dally, William J, and Keutzer, Kurt. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Kim, Yong-Deok, Park, Eunhyeok, Yoo, Sungjoo, Choi, Taelim, Yang, Lu, and Shin, Dongjun. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

Lane, Nicholas D, Bhattacharya, Sourav, Georgiev, Petko, Forlivesi, Claudio, Jiao, Lei, Qendro, Lorena, and Kawsar, Fahim. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Proceedings of IPSN*, 2016.

Li, Hao, Kadav, Asim, Durdanovic, Igor, Samet, Hanan, and Graf, Hans Peter. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. In *Proceedings of ICLR*, 2014.

Lin, Y., Zhang, S., and Shanbhag, N. R. Variation-tolerant architectures for convolutional neural networks in the near threshold voltage regime. In *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 17–22, Oct 2016.

Lin, Yingyan, Sakr, Charbel, Kim, Yongjune, and Shanbhag, Naresh. Predictivenet: An energy-efficient convolutional neural network via zero prediction. In *Proceedings of ISCAS*, 2017.

Parashar, Angshuman, Rhu, Minsoo, Mukkara, Anurag, Puglielli, Antonio, Venkatesan, Rangharajan, Khailany, Brucek, Emer, Joel, Keckler, Stephen W, and Dally, William J. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 27–40. ACM, 2017.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Sakr, Charbel, Kim, Yongjune, and Shanbhag, Naresh. Analytical guarantees on numerical precision of deep neural networks. In Precup, Doina and Teh, Yee Whye (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3007–3016, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. Edge computing: Vision and challenges. volume 3, pp. 637–646, Oct 2016.

Srinivas, Suraj and Babu, R. Venkatesh. Data-free parameter pruning for deep neural networks. *CoRR*, abs/1507.06149, 2015.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, Rabinovich, Andrew, et al. Going deeper with convolutions. CVPR, 2015.

Tai, Cheng, Xiao, Tong, Zhang, Yi, Wang, Xiaogang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

Taigman, Yaniv, Yang, Ming, Ranzato, Marc'Aurelio, and Wolf, Lior. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701–1708, 2014.

Ullrich, Karen, Meeds, Edward, and Welling, Max. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

Wang, Zhangyang, Yang, Jianchao, Jin, Hailin, Shechtman, Eli, Agarwala, Aseem, Brandt, Jonathan, and Huang, Thomas S. Deepfont: Identify your font from an image. In *Proceedings of the 23rd ACM international conference on Multimedia*, pp. 451–459. ACM, 2015.

Yang, Tien-Ju, Chen, Yu-Hsin, and Sze, Vivienne. Designing energy-efficient convolutional neural networks using energy-aware pruning. *arXiv preprint*, 2017.

Zagoruyko, Sergey and Komodakis, Nikos. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zha, Hongyuan, He, Xiaofeng, Ding, Chris, Gu, Ming, and Simon, Horst D. Spectral relaxation for k-means clustering. In *Advances in neural information processing systems*, pp. 1057–1064, 2002.